DİFERANSİYEL EVRİM ALGORİTMASI

DİFERANSİYEL EVRİM ALGORİTMASI

Evrim prensibine dayalı teknikler bilinen optimizasyon metotlarının noksanlıklarının üstesinden gelmek için kullanılmaktadır.

Son zamanlarda bu prensibe dayalı tekniklerin hepsini temsilen ortak bir terim olarak evrimsel hesaplama terimi yaygın olarak kullanılmaya başlanmıştır. Bu sınıfa giren algoritmalara örnek olarak genetik algoritmalar, evrimsel programlama (evolutionary programming) evrimsel stratejiler (evolution strategies) ve DEA vs. sayılabilir.

DİFERANSİYEL EVRİM ALGORİTMASI

Bir problemi çözmede kullanılacak herhangi bir evrimsel algoritma aşağıdaki beş elemana ihtiyaç duymaktadır:

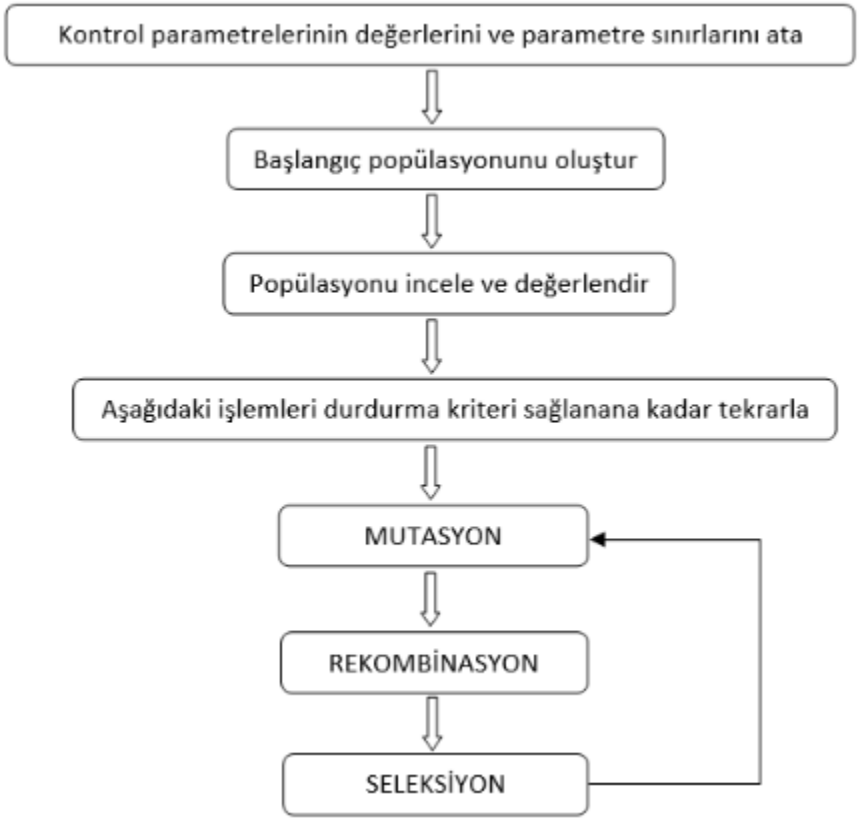
- Problem için çözümlerin genetik temsili.
- Çözümlerin başlangıç popülasyonunu oluşturacak bir yöntem.
- Çözümleri uygunluk açısından değerlendirmeye tabii tutacak değerlendirme fonksiyonu yani çevre.
- Genetik kompozisyonu değiştirecek operatörler.
- Kontrol parametrelerinin değerleri

DİFERANSİYEL EVRİM ALGORİTMASI

- Evrimsel algoritma tek bir bireyle değil bireylerin popülasyonu ile ilgilenir.
- Her birey mevcut problem için muhtemel bir çözümü temsil eder ve bir veri yapısı olarak tanımlanır.
- Diferansiyel Gelişim Algoritması (DGA), popülasyon tabanlı sezgisel bir optimizasyon tekniğidir.
- Global optimizasyon için basit ama güçlü bir tekniktir.
- Özellikle sürekli verilerin söz konusu olduğu problemlere yönelik olarak geliştirilmiştir ve rastlantısal bir yapıya sahiptir.

DİFERANSİYEL EVRİM ALGORİTMASI

- Temeli genetik algoritmaya dayanır. Benzer operatörleri kullansalar da bu operatörlerin yapısı ve kullanım şekilleri birbirinden farklıdır. Genel anlamda algoritmanın adımları aşağıdaki gibidir.



1. Kontrol Parametrelerinin Değerlerini ve Sınırlarını Atama

Temel anlamda parametrelerimiz aşağıdaki gibidir;

•**Değişken Sayısı (D) :** minimum 1 olmak üzere istenilen sayı verilebilir. Kromozomdaki gen sayısı olarak düşünülebilir. (1,2,…….,j)

•**Maksimum Jenerasyon Sayısı (G_{max}) :** Oluşturulacak olan popülasyonun kaç jenerasyon boyunca mutasyon, rekombinasyon ve seleksiyona tabî tutulacağını belirtir. (1,2,…….G_{max})

•**Popülasyon büyüklüğü (NP) :** Popülasyonun büyüklüğünün ne kadar olacağını belirtir. Bu da kromozom sayısı olarak düşünülebilir.

•**Ölçekleme Faktörü (F) :** Ölçekleme faktörüdür, kullanıcı belirleyecektir.

•**Çaprazlama Oranı (CR) :** Olasılığı temsil eder bu yüzden 0 ile 1 arasında değer alır.

•**X^{lo} ve X^{hi}** değerleri ise parametre sınırlarını belirtmektedir.

Kontrol parametrelerinin değerlerinin atanması işlemi şu şekilde temsil edilir;

$$D, G_{max}, NP > 3, F \in (0,1), CR \in [0,1], x^{lo}, x^{hi}$$

2. Başlangıç Popülasyonunun Oluşturulması

Mevcut popülasyon gelecek jenerasyonu oluşturmak için kullanılacağından dolayı başlangıç popülasyonu iyi tanımlanmış sınırlamalara sahip bir araştırma uzayından uniform dağılımla rastgele elemanlar seçilerek oluşturulur.

$$\forall i \leq NP \wedge \forall j \leq D: x_{j,i,G=0} = x_j^{lo} + rand_j\left(x_j^{hi} - x_j^{lo}\right)$$

$$i = (1,2 \ldots\ldots, NP), j = (1,2,\ldots, D), G = 0, randj[0,1] \in [0,1]$$

3. Değerlendirme Fonksiyonu

Yeni bir birey üretildiğinde popülasyona girip girmeyeceğine değerlendirme fonksiyonuyla karar verilir. Hedef kromozomumuzun uygunluk değeri bilinmektedir. Yeni birey onunla karşılaştırılır ve popülasyona girip girmeyeceğine karar verilir. Böylelikle yeterince uygun olmayan bireyler popülasyona eklenmeyerek popülasyonun bozulmaması sağlanır.

4. Mutasyon İşlemi

Mutasyon işlemi parametre optimizasyonu açısından büyük önem taşır. Kromozomun genleri üzerinde rasgele değişiklikler yapmaktır. Mutasyon sayesinde çözüm noktası çözüm uzayında hareket etmektedir. Bir anlamda mutasyon arama uzayını genişletmektedir. Böylelikle daha çok bölgeye ulaşılıp farklı çözümlere ulaşılabilme ihtimali ortaya çıkar. Mutasyondan beklenen çözümün, çözüm uzayında doğru yönde ve doğru miktarda hareket etmesidir. Bu sayede bir yerde takılıp kalınmaz. Fakat bu hareketin beklenen şekilde doğru yönde ve doğru miktarda olması garanti edilemez. Mutasyona tabî tutulacak kromozom dışında birbirinden farklı üç kromozom seçilir. İlk ikisinin farkı alınır ve ölçekleme faktörüyle çarpılır. F genellikle 0 ile 2 arasında değerler almaktadır. Ağırlıklandırılmış fark kromozomu ile üçüncü kromozom toplanır. Amacı amaç vektörleri doğru yönde hareket ettirecek artışları üretmektir.

$$\forall j \leq D: n_{j,i,G+1} = x_{j,r_3,G} + F * \left(x_{j,r_1,G} - x_{j,r_2,G}\right)$$

5. Rekombinasyon İşlemi

Mutasyon işlemi temel olarak yeni araştırma bölgelerinin araştırılmasından sorumludur. Rekombinasyon ya da çaprazlama işleminin amacı ise var olan amaç vektör parametrelerinden yararlanarak yeni vektörler oluşturmak suretiyle araştırmanın başarılı olması için yardımcı olmaktır. Rekombinasyon önceki jenerasyonla başarılı çözümleri birleştirir. Daha iyi bireyler üretmek için eldeki bireyler çaprazlanır. Mutasyon işlemini gerçekleştirirken elde ettiğimiz fark kromozomu ve Xi,g kromozomu kullanılarak yeni deneme kromozomu (Ui,G+1) üretilir. Deneme kromozomuna genler CR olasılıkla fark kromozomundan 1-CR olasılıkla mevcut kromozomdan seçilir. j=jrand koşulu, en az bir tane genin üretilen yeni kromozomdan alınmasının garanti olması için kullanılır.

$$\forall j \leq D, u_{j,i,G+1} = \begin{cases} x_{j,r_3,G}, & eğer\ rand_j[0,1] < CR\ ya\ da\ j = jrand \\ x_{j,i,G}, & diğer \end{cases}$$

6. Seleksiyon İşlemi

Yeni üretilen bireylerin hangi şartlar altında popülasyona girebileceğini tanımlayan kriterdir. Buradaki seleksiyon yöntemi aç gözlü olarak seçilmiştir. Bilindiği üzere diğer evrimsel algoritmalarda kullanılan Deterministik, Rulet Tekerleği, Rasgele ve Turnuva gibi seleksiyon metotları bulunmaktadır. Çalışılan probleme göre bu metotlardan biri tercih edilebilir. Deterministik te belirli sayıdaki en iyi bireyler ile yeni popülasyon oluşturulur, kötü bireyler popülasyona katılmaz. Rulet tekerleğinde, her bireyin çözüme uygunluk derecesi arttıkça yeni popülasyona aktarılma şansı artar. Rasgele seçimde bireylerin uygunluk dereceleri seçilme şanslarını etkilemez. Turnuva metodunda, rasgele seçilen iki birey arasında uygunluk derecesi yüksek olan popülasyona katılırken diğer birey popülasyona kabul edilmez.

$$x_{i,G+1} = \begin{cases} u_{i,G+1}, & eğer\ f(u_{i,G+1}) \leq f(x_{i,G}) \\ x_{i,G}, & diğer \end{cases}$$

Mutasyon, rekombinasyon ve seleksiyon işlemleri durdurma kriteri sağlanana kadar tekrar sırasıyla devam ettirilir. Diferansiyel gelişim algoritmasını maksimum jenerasyon sayısına ulaşana denk çalışır ve ulaştığında algoritma durdurulur. Buna ek olarak algoritmanın çalışması maksimum jenerasyon sayısı ile değil belirlenen durdurma kriterine göre de belirlenebilir. Bu durumda en iyi ve en kötü bireylerin arasındaki farkın belirlenen miktara kadar düşmesi durumunda algoritma sonlandırılır.

$$\left\{ fmax - f(min) \leq \alpha \right.$$

Diferansiyel Gelişim Algoritması sürekli parametreleri söz konusu olduğu durumlarda oldukça başarılı sonuçlar veren bir algoritmadır.

DGA - Özet

- Diferansiyel gelişim algoritması, **aday çözümlerden oluşan bir popülasyon oluşturur.**
- **İteratif olarak** popülasyon üzerinde **birleştirme**, **değerlendirme** ve **seçme işlemlerini yapar.**
- **Birleştirme** yaklaşımı ile **popülasyondan rastgele seçilen iki birey arasındaki farkın** ağırlığına göre **üçüncü bir bireyle birleştirme yapar.**
- Bu yaklaşım ile popülasyondaki **bireylerin çeşitliliğinin artırılması amaçlanmıştır.**
- Diferansiyel gelişim algoritması, **nonlineer** ve **türevlenemez sürekli fonksiyonların optimizasyonu için tasarlanmıştır.**

DGA - Özet

- Ağırlık faktörü $F \in [0, 2]$ **arasındadır** (fark değişimi katsayısı).
- **Ağırlık faktörü** genellikle **0,8 seçilir.**
- **Crossover** ağırlığı $CR \in [0, 1]$ **arasında rastgele seçilir.**
- **Crossover** ağırlığı genellikle **0,9 seçilir.**
- **Başlangıç popülasyonu** tamamen **rastgele oluşturulur.**
- Algoritma kabul edilen konfigürasyonu tanımlayan özel bir **terminolojiye sahiptir (DE/x/y/z).**
- **x çeşitlendirilecek çözümü** (rastgele veya en iyi alınır), **y ise x üzerinde kullanılacak fark vektörlerini**, **z birleştirme operatörünü** (binomial veya exponential) gösterir.
- Popüler konfigürasyonlar:
- **DE/rand/1/***
- **DE/best/2/***

DGA-Özet

- DE/rand/1/bin için pseudocode.

Pseudocode for Differential Evolution

Input: $Population_{size}$, $Problem_{size}$, $Weighting_{factor}$, $Crossover_{rate}$

Output: S_{best}

```
1 Population ← InitializePopulation(Population_size, Problem_size);
2 EvaluatePopulation(Population);
3 S_best ← GetBestSolution(Population);
```

DGA-Özet

```
4  while ¬ StopCondition() do
5  |   NewPopulation ← ∅;
6  |   foreach P_i ∈ Population do
7  |   |   S_i ← NewSample(P_i, Population, Problem_size,
   |   |   Weighting_factor, Crossover_rate);
8  |   |   if Cost(S_i) ≤ Cost(P_i) then
9  |   |   |   NewPopulation ← S_i;
10 |   |   else
11 |   |   |   NewPopulation ← P_i;
12 |   |   end
13 |   end
14 |   Population ← NewPopulation;
15 |   EvaluatePopulation(Population);
16 |   S_best ← GetBestSolution(Population);
17 end
18 return S_best;
```

DGA - Özet

Pseudocode for the `NewSample` function

Input: P_0, Population, NP, F, CR
Output: S

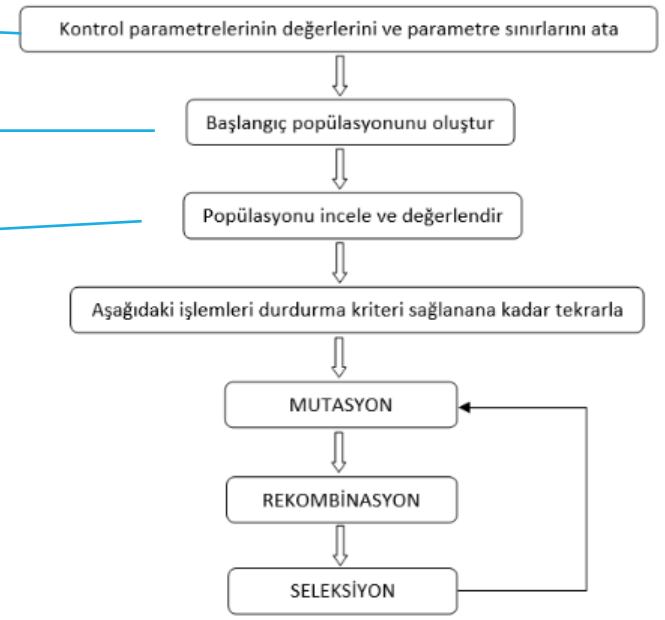
```
1 repeat
2 |   P1 ← RandomMember(Population);
3 until P1 ≠ P0 ;
4 repeat
5 |   P2 ← RandomMember(Population);
6 until P2 ≠ P0 ∨ P2 ≠ P1 ;
7 repeat
8 |   P3 ← RandomMember(Population);
9 until P3 ≠ P0 ∨ P3 ≠ P1 ∨ P3 ≠ P2 ;
```

DGA-Özet

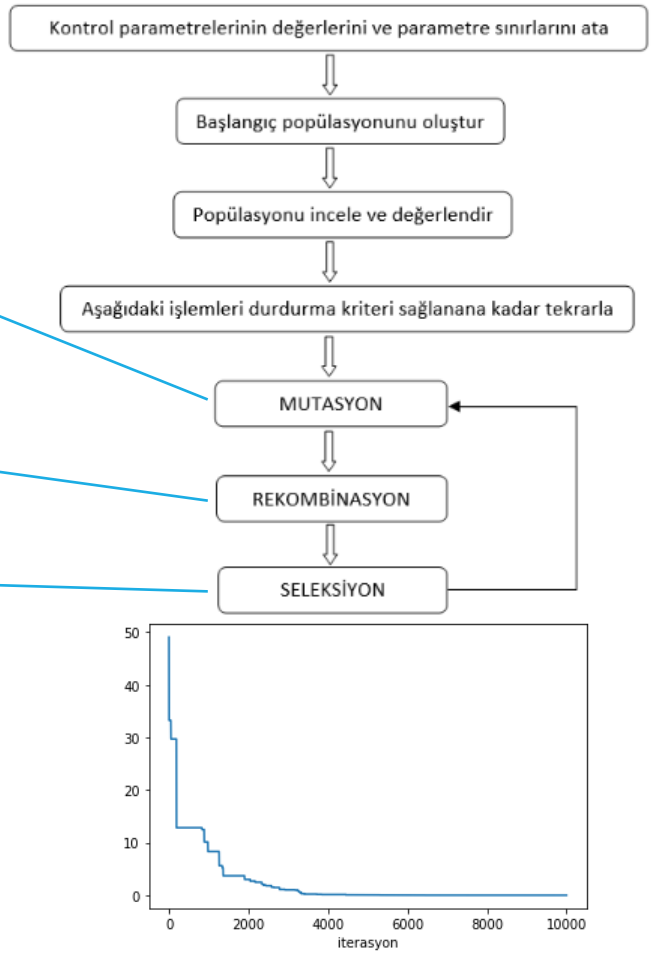
```
10 CutPoint ← RandomPosition(NP);
11 S ← 0;
12 for i to NP do
13 |   if i ≡ CutPoint ∧ Rand() < CR then
14 |   |   S_i ← P_{3_i} + F × (P_{1_i} - P_{2_i});
15 |   else
16 |   |   S_i ← P_{0_i};
17 |   end
18 end
19 return S;
```

```python
import numpy as np
import matplotlib.pyplot as plt

def fun(x):
    return sum(np.power(x,2))

alt=-5
ust=5
D=10
F=0.5
CR=0.5
popSize=100
max_iter=100

populasyon = np.random.ranf([popSize,D]) * (ust-alt) + alt
cocuk = populasyon.copy()
obj = np.zeros(popSize)

for i in range(popSize):
    obj[i]=fun(populasyon[i,:])

best_ind = int(np.round(1+(popSize-1)*np.random.ranf()))
objit = list()
objit.append(obj[best_ind])

```

Sphere fonksiyonu



```python
for iterasyon in range(max_iter):
    for k in range(popSize):
        mut_deg = np.random.permutation(popSize)[:3]
        mutasyon = F*(populasyon[mut_deg[0],:] - \
                    populasyon[mut_deg[1],:]) + \
                    populasyon[mut_deg[2],:]

        for i in range(D):
            if mutasyon[i]>ust:
                mutasyon[i]=ust
            elif mutasyon[i]<alt:
                mutasyon[i]=alt

        for i in range(D):
            secim = np.random.ranf()
            if secim > CR:
                cocuk[k,i] = populasyon[k,i]
            else:
                cocuk[k,i] = mutasyon[i]

        c_obj = fun(cocuk[k,:])
        if c_obj < obj[k]:
            populasyon[k,:] = cocuk[k,:]
            obj[k] = c_obj


        if min(obj) < obj[best_ind]:
            bestVal = min(obj)
            idx = np.where(obj==bestVal)[0][0]
            bestVal = fun(populasyon[idx,:])

        objit.append(bestVal)

print(bestVal)
plt.plot(objit)
plt.xlabel("iterasyon")
plt.show()
```



Evrimsel Programlama

- Evrimsel programlama, **doğal seleksiyon teorisinden esinlenerek geliştirilmiştir.**
- **Bir türün popülasyonu tekrar üretilir** ve **küçük fenotipik değişkenliği olan nesiller oluşturur.**
- **Yeni nesil** ve **ebeveynleri probleme daha uygun olabilmek için yarışır.**
- Problemin **çözümüne daha uygun bireyler sonraki jenerasyona aktarılır.**

Evrimsel Programlama

Algorithm 3.6.1: Pseudocode for Evolutionary Programming.

Input: $Population_{size}$, ProblemSize, BoutSize
Output: S_{best}

```
1 Population ← InitializePopulation(Population_size, ProblemSize);
2 EvaluatePopulation(Population);
3 S_best ← GetBestSolution(Population);
4 while ¬StopCondition() do
5 |   Children ← ∅;
6 |   foreach Parent_i ∈ Population do
7 |   |   Child_i ← Mutate(Parent_i);
8 |   |   Children ← Child_i;
9 |   end
```

Evrimsel Programlama

```
10      EvaluatePopulation(Children);
11      S_best ← GetBestSolution(Children, S_best);
12      Union ← Population + Children;
13      foreach S_i ∈ Union do
14          for 1 to BoutSize do
15              S_j ← RandomSelection(Union);
16              if Cost(S_i) < Cost(S_j) then
17                  Si_wins ← Si_wins + 1;
18              end
19          end
20      end
21      Population ← SelectBestByWins(Union, Population_size);
22  end
23  return S_best;
```