KARADENİZ TEKNİK ÜNİVERSİTESİ
OF TEKNOLOJİ FAKÜLTESİ
YAZILIM MÜHENDİSLİĞİ BÖLÜMÜ

YZM 4032
Meta-Sezgisel Optimizasyon Dersi Dönem Raporu

MPSO Algoritmasının FDB Yöntemi İle İyileştirilmesi

Betül ÇALIK

2020 -2021 BAHAR DÖNEMİ

Adaptif Strateji Kullanan Düzenlenmiş Parçacık Sürü Optimizasyon Algoritmasının FDB Yöntemi ile İyileştirilmesi : Mühendislik Tasarım Problemleri Üzerine Kapsamlı Bir Araştırma

ÖZET

Parçacık Sürü Optimizasyonu (PSO), problem çözmede okul veya kuş sürülerinin hareketlerini simüle eden, iyi bilinen bir sürü zeka tabanlı algoritmadır. Bu makale çalışmasında PSO algoritmasının düzenlenmiş bir versiyonu olan Adaptif Strateji Kullanan MPSO algoritmasının iyileştirilmiş bir versiyonu geliştirilmektedir. MPSO algoritmasında parçacıkların hedef konuma en etkili şekilde ulaşabilmesini modelleyebilmek adına arama sürecine rehberlik eden çözüm adayları uzaklık-uygunluk dengesi (fitness-distance balance, FDB) yöntemi kullanılarak belirlenmiştir. Geliştirilen FDB-tabanlı MPSO algoritmasının performansını test etmek ve doğrulamak amacıyla güncel bir karşılaştırma problemleri havuzu olan CEC 2020 kullanılmıştır. Önerilen algoritmanın farklı tiplerdeki ve farklı boyutlardaki arama uzaylarındaki performansını test etmek ve doğrulamak için test problemleri 3/50/100 boyutta tasarlanmıştır.

Anahtar Kelimeler: Parçacık sürü optimizasyon algoritması, düzenlenmiş parçacık sürü optimizasyon algoritması, uygunluk-mesafe dengesi seçim yöntemi, meta-sezgisel optimizasyon

1. GİRİŞ

Uzman sistemlerde herhangi bir problem; güç sistemi, yol planlama ve su tedarik sistemi tasarımı, operasyon yönetimi ve uydu ağı gibi optimizasyon yöntemleri olarak modellenebilir. Uzman sistem uygulamalarının oldukça geniş bir kullanım alanına sahip olduğu için karmaşık optimizasyon problemleri genellikle doğrusal olmayan, belirsiz, çok modlu, süreksiz ve yüksek boyutludur. Bu durum, yüksek performanslı bir optimizasyon algoritması geliştirmeyi etkin bir araştırma noktası haline getirmiştir.

PSO algoritması, araştırmacılardan giderek daha fazla ilgi gören ve uzman sistemlerde birçok gerçek optimizasyon problemini çözmek için başarıyla uygulanan (Kennedy & Eberhart, 1995) tarafından önerilen popülasyon tabanlı bir optimizasyon algoritmasıdır. Araç yönlendirme problemi, veri sınıflandırması, özellik seçimi, derin sinir ağı ve birçok gerçek problemin çözümünde kullanılmıştır.

PSO algoritması basit prensibine, kolay uygulamasına ve yüksek yakınsama oranına sahip olmasına rağmen erken yakınsama ve zayıf global arama yeteneği gibi bazı eksikliklere sahiptir. Araştırmacılar bu eksikliklerin üstesinden gelmek için PSO'nun çeşitli değiştirilmiş sürümlerini önermişlerdir.

Her ne kadar verilen PSO varyantları olgunlaşmış ve uzman sistemlerde gerçek optimizasyon problemlerini çözmek için başarılı bir şekilde uygulanmış olsa da, optimizasyon problemlerinde erken yakınsama ve zayıf performans problemleri hala mevcuttur. Ek olarak, insan toplumunun gelişmesiyle birlikte optimizasyon problemleri gittikçe karmaşıklaşmaya ve yüksek boyutlu, çok kısıtlamalı ve çok modlu olma eğiliminde olmaya devam etmektedir. Bu durum PSO'nun nasıl iyileştirileceği konusunda araştırmaların artmasını sağlamıştır.

MPSO algoritması, PSO'nun performansını güçlendirmek için modifiye edilmiştir. PSO algoritmasına ek olarak MPSO'da aşağıda açıklanan dört katkı vardır.

1) Arama adımını genişleterek veya daraltarak keşif kullanımını daha iyi dengelemek için kaos tabanlı doğrusal olmayan bir inertia weight önerilmiştir.
2) Stochastic ve mainstream öğrenme teknikleri, popülasyonun çeşitliliğini etkin bir şekilde arttırabilen ve erken yakınsamayı önleyebilen bir şekilde tasarlanmıştır. Ardından, algoritmanın uzman sistemlerde karmaşık optimizasyon problemlerini çözme yeteneği geliştirilmiştir.
3) Keşif ve kullanım sürecini dengelemek için adaptif bir konum güncelleme stratejisi geliştirilmiştir.
4) Doğadaki en güçlü olanın hayatta kalması kuralından esinlenilerek MPSO'nun yakınsama kesinliğini arttırmak için bir terminal değiştirme mekanizması benimsenmiştir. Böylece algoritmanın gerçek optimizasyon problemlerini çözme yeteneği daha da geliştirilmiştir.

2. YÖNTEM

MSA algoritmaları optimizasyon sürecinde esas olarak iki gereksinimi yerine getirmektedirler. Bunlar; arama uzayındaki referans bir konumun yakın komşuluğunda hassas bir şekilde araştırma yapabilmek ve ihtiyaç duyulduğunda arama uzayındaki umut vadeden konumları etkili bir şekilde keşfedebilmektir. MSA algoritmaları bu gereksinimleri yerine getirirken benzer adımlardan oluşan bir arama süreci yaşam döngüsünü takip ederler. Buna göre MSA algoritmalarında arama sürecinin genel adımları Algoritma 1'de verilmektedir.

Algoritma 1. MSA algoritmalarında arama sürecinin genel adımları [22]

1) Optimizasyon probleminin tanımlanması (amaç fonksiyonu, tasarım parametreleri vd. öğeler)
2) Algoritma parametrelerinin tanımlanması ve çözüm adayları topluluğunun oluşturulması
3) Aday çözümlerin uygunluk değerlerinin hesaplanması
4) Arama Süreci Yaşam Döngüsü
    a. Seçim süreci
    b. Komşuluk araması ve çeşitliliğin sağlanması
    c. Çözüm adayı topluluğunun güncellenmesi
5) Sonlandırma kriteri sağlandı mı?
    a. Hayır (Adım 4'e dön)
    b. Evet (Arama sürecini sonlandır ve en iyi çözüm adayını kaydet)

Algoritma 1'de verilen adımlara göre MSA algoritmalarında ilk üç adım ortaktır. Algoritmaların performansları arasındaki farklılıklar ise dördüncü adımdaki işlemlere bağlıdır. Dördüncü adımda her algoritmanın kendine özgü arama süreci yaşam döngüsü işletilmektedir. Arama süreci yaşam döngüsünün ilk adımında arama sürecine rehberlik edecek olan adaylar seçilir. Seçim süreci popülasyon üyeleri arasından rastgele, aç gözlü, olasılıksal yöntemlerden biri kullanılarak gerçekleştirilebilir. Rastgele seçim yöntemi arama sürecinde çeşitliliğe katkı sunmak amacıyla kullanılır. Ancak özellikle büyük boyutlu ve karmaşıklık düzeyi yüksek olan arama uzaylarında etkisi düşüktür. Aç gözlü yöntem ile popülasyon üyeleri arasından en başarılı olanın seçilir. Başarı ölçütü ise çözüm adaylarının uygunluk değerleridir. Örneğin ABC'de elit arılar , ALO'da elit karınca , ASO'da en iyi parçacık , EFO'da pozitif yüklü parçacıklar, COA'da alpha birey, GWO'da alfa, beta, gama bireyleri açgözlü yöntem kullanılarak seçilirler. Olasılıksal seçim yöntemi, rastgele ve açgözlü yöntemlerin karmasıdır. Çözüm adaylarının uygunluk değerleri ile doğru orantılı olacak şekilde seçilme olasılıkları belirlenir. Olasılıksal seçim yönteminin en yaygın bilinen iki tekniği rulet ve ikili turnuvadır. Rulet tekerleğinin her bir parçası temsil ettiği bireyin uygunluk değeri ile doğru orantılı olacak şekilde tasarlanır. Tekerlek üzerinden rastgele seçilen parça hangi bireye aitse o seçilmiş olur. İkili turnuva tekniğinde ise popülasyondan rastgele seçilen iki birey arasından uygunluk değeri büyük olan seçilir. Olasılıksal seçim yöntemi Genetik Algoritma'da ve bu evrimsel esaslı algoritmaların varyasyonlarında

ebeveyn seçiminde kullanılır. MSA algoritmalarında seçim sürecinden sonra komşuluk araması ve çeşitlilik işlemleri yerine getirilir. Seçim sürecinde belirlenen bireyler referans (rehber) alınarak hassas ve etkili bir arama gerçekleştirilmeye çalışılır. Bu aşamanın başarısı arama operatörlerinin yeteneklerine bağlı olduğu kadar referans konumların doğru seçilmesine de bağlıdır. Dolayısıyla seçim sürecinin başarısı, MSA algoritmalarının nihai başarıları üzerinde oldukça etkilidir. MSA algoritmalarında arama süreci yaşam döngüsünün son adımı çözüm adayları topluluğunun güncellenmesidir. MSA'larında popülasyon büyüklüğü sabittir. Bundan dolayı arama sürecinde popülasyona eklenen her yeni bireyin yerine popülasyondan bir birey çıkarılmak durumundadır. Bireylerden hangisinin popülasyonda kalıp hangisinin ayrılacağına karar vermek için ise istisnalar olmakla birlikte aç gözlü yöntem kullanılarak karar verilir. Popülasyonda kalacak olan bireye karar verilirken evrim yasası işletilerek "güçlü olan hayatta kalır" ilkesine paralel olarak uygunluk değeri büyük olan birey popülasyonda kalır. MSA algoritmalarında son adım arama süreci yaşam döngüsünün sonlandırılmasıdır. MSA'larının arama süreci yaşam döngülerinin her bir adımında amaç fonksiyonlarını çağırma sayıları değişebilmektedir. Örneğin ABC algoritmasında bu sayı kolonideki takipçi/izci/kâşif arılar kadarken, SOS algoritmasında dört, GA'da ebeveyn sayısı kadar (genellikle iki), HHO'da bir ve MRFO'da popülasyondaki birey sayısı kadardır. Dolayısıyla MSA'larında arama sürecini sonlandırmak amacıyla iterasyon sayısını kullanmak algoritmaların eşit şartlarda arama yapmaları konusunda adaletsizliğe neden olur. MSA'larından fırsat eşitliğini sağlamanın yolu her algoritma için amaç fonksiyonunu azami çağırma sayısını eşit tutmaktır. MSA'ları konusunda 90'lı yıllardan bu yana düzenli bir şekilde yürütülen CEC konferanslarında arama süreci sonlandırma kriteri olarak genellikle amaç fonksiyonunu çağırma sayısı 10.000*d (problem boyutunun on bin katı) olarak tanımlanmaktadır.

2.1. Adaptif Strateji Kullanan MPSO Algoritması

PSO'nun erken yakınsama fenomeninden kaçınmak ve karmaşık problemlerde performansını iyileştirmek için MPSO algoritması önerilmiştir. MPSO algoritmasının dört ana vurgusu aşağıda ele alınmıştır.

1) Chaotic-Based Inertia Weight

PSO'daki parametrelerden biri olan inertia weight (ω), parçacıklara farklı ortamlarda dinamik ayarlama kabiliyeti sağlayabilir. Böylece keşif ve sömürü arasındaki dengeyi sağlayabilir, bu nedenle PSO'da önemli bir rolü bulunmaktadır. Genel olarak doğrusal

inertial weight yöntemi benimsenir, fakat doğrusal olmayan inertial weight daha güçlü uyum ve simülasyon kabiliyetine sahiptir.

Doğrusal olmayan bir haritalama olarak, kaos iyi rastgelelik ve düzensizliğe sahip rastgele sayılar üretir, bu nedenle evrimsel hesaplama alanında yaygın olarak kullanılmaktadır. Lojistik, 0 ile 1 arasında rastgele sayılar üretebilen iyi bilinen bir kaotik haritalamadır. Denklem 1'de tanımlanmıştır. Lojistik kaos, inertia weight'e dahil edilerek denklem 2 ile tanımlanan lineer olmayan bir inertia weight oluşturulmuştur.

$r\,(t+1) = 4r(t)(1-r(t)) \qquad r(0) = rand$	**(1)**
Where $r_0 \notin \{0,0.25,0.5,0.75,1\}$ $\omega(t) = r(t)\cdot\omega\min + \frac{(\omega\max - \omega\min)\,.\,t}{Tmax}$	**(2)**

ωmax = 0.9, ωmin = 0.4 olduğunda üretilen sayıdır. R(t) ise lojistik kaotik tarafından üretilen rastgele sayıdır.

2) Stochastic ve Mainstream Öğrenme Stratejileri

Kişisel ve global öğrenme stratejileri PSO'da benimsendiği için, parçacıklar hızlarını ve konumlarını güncellemek için Pbest (kişisel en iyi) ve Gbest'ten (global en iyi) öğrenirler. Bu öğrenme stratejisi PSO'nun hızlı yakınsama, yüksek güvenilirlik avantajlarına sahip olmasını sağlamaktadır. Ancak yine bu strateji, erken yakınsama ve karmaşık problemlerde düşük performans gibi eksikliklere yol açar. Bu sorunlardan kaçınmak için kişisel ve global öğrenme stratejilerinin yerini alacak olan stochastic ve mainstream öğrenme stratejileri kullanılmıştır.

Stochastic öğrenme, parçacıklara popülasyondaki diğer mükemmel bireylerden öğrenme yeteneği verir ve parçacıkların hareketlerini daha çok çeşitlendirir. Bu nedenle PSO algoritmasının erken yakınsama problemi, popülasyon çeşitliliği arttırılarak aşılmaktadır.

Özetle stochastic ve mainstream öğrenme stratejileri önerilmiştir ve orijinal halindeki kişisel ve global öğrenme stratejileri değiştirilmiştir. Bu nedenle hız güncelleme denklemi aşağıdaki gibi verilmiştir.

$$Vi(t+1) = \omega(t)Vi(t) + r1c1 \otimes (SPbesti(t) - Xi(t)) + r2c2 \otimes (Mbest(t) - Xi(t))$$	**(3)**

Denklem 3'ün sağ tarafındaki ilk bileşen inertia weight, ikinci bileşen stochastic öğrenme bileşeni ve üçüncü bileşen de mainstream öğrenme bileşeni olarak ayarlanmıştır.

3) Adaptif Pozisyon Güncelleme Stratejisi

Farklı konum güncelleme stratejilerinin farklı keşif ve sömürü yeteneklerine sahip olduğu iyi bilinmektedir. Yerel sömürü ve küresel keşfi dengelemek için, uyarlanabilir bir konum güncelleme mekanizması önerilmektedir. Bu mekanizmayı kullanarak, parçacıklar keşif ve kullanımı daha iyi dengelemek için karşılık gelen koşullara göre konum güncelleme stratejileri seçebilirler. Belirtildiği gibi, uyarlanabilir strateji konum güncelleme stratejisi Denklem 4 ve 5 ile temsil edilmektedir.

$$p_i = \frac{exp(fit(X_i(t)))}{exp(\frac{1}{N}\sum_{i=1}^{N} fit(X_i(t)))}$$	**(4)**
$$Xi(t+1) = \begin{cases} \omega(t)Xi(t) + (1-\omega(t))Vi(t+1) + Gbest(t) & pi > rand \\ Xi(t) + Vi(t+1) & otherwise \end{cases}$$	**(5)**

Denklem 5'e göre, pi küçük olduğunda, i parçacığının performansı ortalama popülasyon seviyesinden daha yüksektir, Bu durumda, küresel keşif kabiliyetini geliştirmek için X = X + V stratejisi konumu güncellemek için uyarlanmıştır. Aksine, pi daha büyük olduğunda, parçacık i'nin ortalama nüfus seviyesinden daha zayıf olduğu kabul edilir ve konum güncelleme stratejisi X = ωX + (1 − ω) V, lokal kullanım kabiliyetini geliştirmek için benimsenir.

4) Terminal Değiştirme Mekanizması

Popülasyon çeşitliliğini arttırmak ve MPSO'nun karmaşık problemlerdeki performansını iyileştirmek için terminal değiştirme mekanizması kullanılmaktadır. Her iterasyonda, global en kötü parçacık olan Gworst değiştirilecektir. İlk olarak en kötü parçacık Denklem 5'te tanımlanmıştır. Daha sonra denklem 6'da çaprazlama ile Nbest üretilmiştir. Son olarak Gworst ile Nbest karşılaştırılarak yeni üretilen Nbest Gworst'tan daha iyiyse, o zaman Gworst'un ilgili kişisel en iyisi Nbest ile değiştirilecek, aksi takdirde, Gworst'un karşılık gelen kişisel en iyisi değişmeden kalacaktır. Bu değiştirme mekanizması Denklem 8 olarak tanımlanmıştır.

$Gworst(t) = max\{Pbest_1(t), Pbest_2(t), \cdots, Pbest_N(t)\}$	**(6)**
$Nbest(t) = Gbest(t) + rand \cdot (Pbest_j(t) - Pbest_k(t))\ j \neq k$	**(7)**
$Gworst(t) = \begin{cases} Nbest(t) & fit(Nbest(t)) < fit(Gworst(t)) \\ Gworst(t) & otherwise \end{cases}$	**(8)**

Özet olarak MPSO algoritması PSO'nun bir varyantı olarak önerilmiştir ve pseudocode'u aşağıdaki gibi verilmiştir.

Algoritma 1. MPSO Algoritması Sözde Kodu

i) Gereksinimler:
- P: Popülasyon
- N: Popülasyonun boyutu
- T: Komşu parçacıkların sayısı
- T_{max}: Maksimum iterasyon sayısı

ii) Popülasyonun başlatılması:
- Popülasyondaki her bir parçacık için hız ve konum vektörleri oluşturulur ve uygunluk değerleri hesaplanır.

iii) Arama süreci yaşam döngüsü:
While $t <= t_{max}$
Her bir vektöre ait kişisel (Pbest) ve global (Gbest) konumlar adaptif olarak güncellenir
Kişisel ve global konumlar fitness değerine göre güncellenir (terminal değiştirme mekanizması)
end

2.2. FDB Yöntemi

FDB, MSA algoritmalarının tasarımı için geliştirilmiş ve arama sürecinde popülasyonundaki en iyi çözüm adayına en fazla katkıyı sağlayabilecek olanın etkili bir şekilde belirlenmesini sağlayan bir seçim yöntemidir. MHS algoritmalarında topluluk içindeki bireyler için FDB seçim yöntemi kullanılarak çözüm adaylarının arama sürecine katkılarına işaret eden skor değerleri hesaplanmaktadır. Skor hesabında bireylerin iki özelliği dikkate alınmaktadır. Bunlar, uygunluk değerleri ve popülasyonda anlık olarak en iyi çözüm adayına olan uzaklık değerleridir. Buna göre, m-adet amaç fonksiyonundan, J+K adet kısıttan oluşan bir kısıtlı optimizasyon problemi Eşitlik (9)'da verildiği gibi temsil edilir.

$$\underset{x \in R^n}{minimize\ /\ maximize}\ G = f(x_1, x_2, \ldots, x_m) \quad \text{Amaç}$$
$$\emptyset_j(x) = 0, (j = 1, 2, \ldots, J), \tag{9}$$
$$\varphi_k(x) \leq 0, (k = 1, 2, \ldots, K)$$

Eşitlik 9'da $\emptyset_j$ ve φ_k sırasıyla eşitlik ve eşitsizlik kısıtlarını temsil ederler. Eşitlik 9'da verilen bir optimizasyon probleminin çözümlenmesi için geliştirilmiş bir MSA algoritmasının n-adet çözüm adayından oluştuğu kabul edilsin. Buna göre çözüm adayları vektörü (popülasyon, *P*) ve bu adayların uygunluk değerleri vektörü (*F*) Eşitlik (10)'da verildiği gibi temsil edilebilir.

, $$P \equiv \begin{bmatrix} p_1 \\ \vdots \\ p_n \end{bmatrix} \equiv \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{bmatrix}_{nxm} \qquad F \equiv \begin{bmatrix} f_1 \\ . \\ . \\ . \\ f_n \end{bmatrix}_{nx1} \tag{10}$$

Yukarıda verilen tanımlamalara göre çözüm adaylarının FDB skorlarının hesaplanma adımları aşağıda verilmektedir.

i) Popülasyondaki *i*-inci çözüm adayı P_i'nin, popülasyonun *t*-inci anındaki en iyi çözüm adayı olan P_{best}'e olan oklit uzaklığı Eşitlik 11'de verildiği gibi hesaplanır:

$$_{i=1}^{n}\forall\, P_i, D_{P_i} = \sqrt{\left(p_{i[1]} - p_{\text{best}[1]}\right)^2 + \left(p_{i[2]} - p_{\text{best}[2]}\right)^2 + \cdots + \left(p_{i[m]} - p_{\text{best}[m]}\right)^2} \quad \textbf{(11)}$$

ii) Popülasyondaki bireylerin P_{best}'ten uzaklığını temsil eden D_p vektörü Eşitlik (12)'de verilmiştir.

$$D_P \equiv \begin{bmatrix} d_1 \\ . \\ . \\ . \\ d_n \end{bmatrix}_{nx1} \quad \textbf{(12)}$$

iii) Çözüm adaylarının FDB skorları hesaplanırken Eşitlik 10'da verilen uygunluk değerleri vektörü (F) ve Eşitlik 12'de verilen uzaklık değerleri vektörü D_p kullanılır. Bu iki parametrenin skor hesaplamasında birbirlerine hakim olmaması için normalleştirilmesi gerekir. Buna göre, her iki parametrenin [0, 1] normalize edilmiş uygunluk ve uzaklık değerleri, sırasıyla normF ve normDp ile temsil edilir. Çözüm adaylarının FDB skorları (S_p) Eşitlik 13'te verildiği gibi hesaplanır.

$$_{i=1}^{n}\forall\, P_i,\ S_{P_{[i]}} = w * \ normF_{[i]} + (1 - w) * normD_{P_{[i]}} \quad \textbf{(13)}$$

Eşitlik 13'te w, uygunluk ve uzaklık parametrelerinin FDB skoru üzerindeki etkilerini temsil eden ağırlık katsayısıdır. FDB yönteminin tanıtıldığı çalışmada w=0.5 olarak alınmıştır.

iv) Buna göre, *P*-popülasyonundaki bireylerin FDB skorlarını temsil eden *n*-boyutlu S_p vektörü Eşitlik 14'te verilmektedir.

$$S_P \equiv \begin{bmatrix} s_1 \\ . \\ . \\ . \\ s_n \end{bmatrix}_{nx1} \quad \textbf{(14)}$$

Çözüm adaylarının FDB skorlarını temsil eden S_p oluşturulduktan sonra, MHS algoritmalarında arama sürecine rehberlik edecek çözüm adaylarının seçiminde açgözlü yöntem kullanılarak FDB skoru en yüksek olan birey seçilir.

2.3. MPSO Algoritmasının FDB Yöntemi ile Hesaplanması

Bu bölümde MPSO algoritmasının FDB yöntemi ile tasarlanması süreci tanıtılmaktadır.

Algoritma 2'de MPSO-FDB algoritmasının sözde kodu verilmektedir.

Algoritma 2. MPSO-FDB algoritmasının sözde kodu	
1.	**Başla**
2.	**Parçacık Sürüsü Popülasyonunun Yaratılması**
3.	X: Konum vektörü rastgele oluşturulur.
4.	V: Hız vektörü rastgele oluşturulur.
	X ve V'nin uygunluk değerleri hesaplanır.
5.	**Arama Sürecinin Başlaması**
6.	**While** fes < maxFEs
7.	X ve V güncellenir.
8.	Stokastik ve ana akım öğrenme stratejileri uygulanır.

9.	**Çeşitlilik ve Güncelleme Süreci**
10.	Adaptif konum güncelleme stratejisi uygulanır. **if** rand < 0.4 X(i,:)=w*X(**FdbSeçimYöntemi()**, :)+(1-w)*V(i,:)+Gbest; **else** X(i,:)=w*X(i,:)+(1-w)*V(i,:)+Gbest; **end**

11.	Terminal değiştirme mekanizması uygulanır.
	if rand < 0.8
	fX(i)= testFunction(X(**FdbSeçimYöntemi()**,:)', fhd, fun);
	else
	fX(i)= testFunction(X(i,:)', fhd, fun);
	end

12.	Uygunluk değeri ve kişisel en iyi değer (Pbest) güncellenir. if rand < 0.3 Pbest(i,:)=X(**FdbSeçimYöntemi()**, :); else Pbest(i,:)=X(i,:); end
13.	**end while**
14.	**end**

3. DENEYSEL ÇALIŞMA AYARLARI

3.1. Ayarlar

Deneysel çalışmaları objektif ve adil bir şekilde yürütebilmek için aşağıdaki prosedürler izlenmiştir:

- Deneysel çalışma ayarları için CEC 2020 konferansında tanımlanan şartlar referans alınmıştır.
- MPSO algoritmasının parametrelerinin ayarlanmasında, kendi çalışmasında verilen ayarlar, yani popülasyon boyutu ve diğer ayarları referans olarak alınmıştır.
- Algoritmalar arasında fırsat eşitliğini sağlamak için, amaç fonksiyonunu azami değerlendirme sayısı üzerinden sonlandırma kriteri tanımlanmıştır. Bu değer 10.000*d (d:problem boyutu) dir.
- Deneysel çalışmalar MATLAB®R2016b'de, AMD Ryzen 7 4800H, 2.90GHz ve 16 GB RAM ve x64 tabanlı işlemci üzerinde gerçekleştirildi.

4. ANALİZ SONUÇLARI

1. Deney Sonuçları

Algoritmanın performansında iyileştirme yapmak için FDB yöntemi aşağıdaki ayarlarla birlikte kullanılmıştır.

- Çalışma, CEC 2020 üzerinde 30 boyutlu olacak şekilde yapılmıştır.
- Maksimum iterasyon sayısı 10.000 * D olarak belirlenmiştir.

Case Adı	Boyut / D	Uygulanan Aşama	Uygulanan equation	Uygulanan Yer	Uygulanan Oran	İyi	Aynı	Kötü
MPSO_FDB_case1	30	Aşama 1	1	A	0.4	1	9	0
MPSO_FDB_case2	30	Aşama 1	1	A	0.8	0	10	0
MPSO_FDB_case3	30	Aşama 1	1	B	0.4	0	10	0
MPSO_FDB_case4	30	Aşama 1	1	B	0.8	0	10	0
MPSO_FDB_case5	30	Aşama 1	2	A	0.4	0	10	0
MPSO_FDB_case6	30	Aşama 1	2	B	0.4	1	8	1
MPSO_FDB_case7	30	Aşama 2	3	A	0.8	0	9	1
MPSO_FDB_case8	30	Aşama 2	4	A	0.7	0	10	0
MPSO_FDB_case9	30	Aşama 2	5	A	0.8	0	10	0
MPSO_FDB_case10	30	Aşama 3	6	A	0.3	0	10	0
MPSO_FDB_case11	30	Aşama 3	7	A	0.6	0	10	0
MPSO_FDB_case12	30	Aşama 3	8	A	0.4	0	10	0
MPSO_FDB_case13	30	Aşama 3	9	A	0.7	0	10	0

2. Deney Sonuçları

Algoritmanın performansında iyileştirme yapmak için FDB yöntemi aşağıdaki ayarlarla birlikte kullanılmıştır.

- Çalışma, CEC 2020 üzerinde 50 boyutlu olacak şekilde yapılmıştır.
- Maksimum iterasyon sayısı 10.000 * D olarak belirlenmiştir.

• Case Adı	Boyut / D	Uygulanan Aşama	Uygulanan equation	Uygulanan Yer	Uygulanan Oran	İyi	Aynı	Kötü
MPSO_FDB_case1	30	Aşama 1	1	A	0.4	0	10	0
MPSO_FDB_case2	30	Aşama 1	1	A	0.8	0	10	0
MPSO_FDB_case3	30	Aşama 1	1	B	0.4	0	10	0
MPSO_FDB_case4	30	Aşama 1	1	B	0.8	0	9	1
MPSO_FDB_case5	30	Aşama 1	2	A	0.4	1	9	0
MPSO_FDB_case6	30	Aşama 1	2	B	0.4	0	9	1
MPSO_FDB_case7	30	Aşama 2	3	A	0.8	0	10	0
MPSO_FDB_case8	30	Aşama 2	4	A	0.7	0	10	0
MPSO_FDB_case9	30	Aşama 2	5	A	0.8	0	10	0
MPSO_FDB_case10	30	Aşama 3	6	A	0.3	0	10	0
MPSO_FDB_case11	30	Aşama 3	7	A	0.6	0	10	0
MPSO_FDB_case12	30	Aşama 3	8	A	0.4	0	10	0
MPSO_FDB_case13	30	Aşama 3	9	A	0.7	0	9	1

3. Deney Sonuçları

Algoritmanın performansında iyileştirme yapmak için FDB yöntemi aşağıdaki ayarlarla birlikte kullanılmıştır.

- Çalışma, CEC 2020 üzerinde 100 boyutlu olacak şekilde yapılmıştır.
- Maksimum iterasyon sayısı 10.000 * D olarak belirlenmiştir.

• Case Adı	Boyut / D	Uygulanan Aşama	Uygulanan equation	Uygulanan Yer	Uygulanan Oran	İyi	Aynı	Kötü
MPSO_FDB_case1	30	Aşama 1	1	A	0.4	0	10	0
MPSO_FDB_case2	30	Aşama 1	1	A	0.8	0	10	0
MPSO_FDB_case3	30	Aşama 1	1	B	0.4	0	10	0
MPSO_FDB_case4	30	Aşama 1	1	B	0.8	0	10	0
MPSO_FDB_case5	30	Aşama 1	2	A	0.4	0	10	0
MPSO_FDB_case6	30	Aşama 1	2	B	0.4	1	9	0
MPSO_FDB_case7	30	Aşama 2	3	A	0.8	1	9	0
MPSO_FDB_case8	30	Aşama 2	4	A	0.7	0	10	0
MPSO_FDB_case9	30	Aşama 2	5	A	0.8	0	10	0

MPSO_FDB_case10	30	Aşama 3	6	A	0.3	0	10	0
MPSO_FDB_case11	30	Aşama 3	7	A	0.6	0	10	0
MPSO_FDB_case12	30	Aşama 3	8	A	0.4	1	9	0
MPSO_FDB_case13	30	Aşama 3	9	A	0.7	1	9	0

5. SONUÇLAR

Bu makale çalışmasında güncel bir meta-sezgisel arama algoritması olan MPSO benzer sonuçlarla yeniden tasarlanmıştır. Yeni bir seçim yöntemi olan FDB sayesinde MPSO algoritması taklit ettiği doğal süreçle daha uyumlu olacak şekilde tasarlanabilmiştir.