07.05.2025

Algoritma Tasarımı ve Analizi Ders Projesi

(BİLGİSAYAR MÜHENDİSLİĞİ_YL)

Bald Eagle Search ve Golden Sine Algoritmaları

Yasin Ünal
503020250009

1. Giriş

Meta-sezgisel algoritmalar, karmaşık optimizasyon problemlerini çözmek için doğadan, biyolojiden veya matematiksel kavramlardan ilham alan yöntemlerdir. Bu algoritmalar, genellikle deterministik olmayan yapıları sayesinde global optimum çözümlere ulaşmada etkilidirler. Son yıllarda, bu alanda birçok yeni algoritma geliştirilmiştir.

1.1 Bald Eagle Search Algorithm (BES)

Bald Eagle Search Algorithm (BES), doğadan esinlenen ve bir kartalın avlanma stratejisinden esinlenerek geliştirilen bir optimizasyon algoritmasıdır. **İlk olarak 2019'da Alsattar ve ark. tarafından tanıtılmıştır** [1] . BES, global ve yerel arama mekanizmalarını dengeli bir şekilde kullanarak optimizasyon problemlerini çözmek için tasarlanmıştır. Bu algoritma, literatürde çeşitli alanlarda uygulanmış ve geliştirilmiştir. 2019-2024 yılları arasında yapılan çalışmalarda, BES'nın farklı varyasyonları ve hibrit versiyonları önerilmiştir. Örneğin, **rüzgar enerjisi sistemlerinde maksimum güç takibi** (Fathy et al., 2022), **konum tespiti problemleri** için geliştirilen HBES (Liu et al., 2022), ve global optimizasyon problemleri için önerilen **SABES** (Sharma et al., 2022) dikkat çeken çalışmalardır [2], [3], [4].

Algoritmanın gelişimi devam ederek, çok hedefli optimizasyon problemlerine yönelik MOBES (Zhang et al., 2023), elektromanyetik soğurucuların tasarımı için BESOA (Kankılıç&Karpat, 2023), ve veri kümeleme için BESAVOA (Gharehchopogh, 2024) gibi spesifik uygulamalar geliştirilmiştir [5], [6], [7]. Özellikle **enerji yönetimi alanında**, **akıllı ev sistemlerinin optimizasyonu** için IBES (Youssef et al., 2024) önerilmiştir [8].

El-Shorbagy ve arkadaşlarının 2024'te yayınladığı **kapsamlı literatür taraması**, BES'nin teorik temelleri, varyantları ve uygulama alanlarını detaylı bir şekilde inceleyerek, algoritmanın optimizasyon alanındaki önemini vurgulamıştır [9] . Bu gelişmeler, BES'nin çeşitli mühendislik ve bilimsel problemlerin çözümünde etkili bir araç olduğunu göstermektedir.

Kullanım Alanları

- Sürekli optimizasyon problemleri
- Mühendislik tasarımı problemleri
- Sınıflandırma problemlerinde öznitelik seçimi

1.2. Golden Sine Algorithm (GSA)

Golden Sine Algorithm (GSA), trigonometri ve altın oran prensiplerinden esinlenerek geliştirilen bir optimizasyon algoritmasıdır. Algoritma, sinüs fonksiyonlarının özelliklerini kullanarak arama sürecini optimize eder. İlk olarak **Tanyıldızı ve Demir tarafından 2017 yılında geliştirilmiştir** [10]. İlerleyen yıllarda birçok araştırmacı, GSA'yı farklı optimizasyon algoritmalarıyla birleştirerek veya geliştirerek yeni hibrit çözümler ortaya koymuştur [11].

2019'da Xie ve arkadaşları, **Black Hole Algorithm (BHA)** algoritmasını GSA ile birleştirerek **Golden Sine and Black Hole Algorithm (GSLBH)**'yi geliştirmiş, 2020'de Zhang ve Wang ise **Whale Optimization Algorithm (WOA)**'yı GSA ile entegre ederek **New Golden Sine-Whale Optimization Algorithm (NGS-WOA)**'yı ortaya koymuştur [12], [13] . 2021'de Ghanbari ve Goldani, GSA'yı **Support Vector Regression (SVR)** parametrelerinin optimizasyonunda kullanarak finans alanında başarılı sonuçlar elde etmiştir [14].

2022 yılı, GSA tabanlı hibrit algoritmaların yoğun olarak geliştirildiği bir dönem olmuştur. Han ve arkadaşları **Golden Sine and Dynamic Multi-Population Algorithm (GDMPA)**'yı, Liu ve ekibi **Hybrid Adaptive Golden Sine Algorithm (HAGSA)**'yı, Yuan ve arkadaşları ise **Lightweight Golden Sine and Golden Jackal Optimization (LSGJO)**'yu geliştirmiştir [15], [16] . Bu algoritmalar, özellikle mühendislik tasarım problemlerinde ve endüstriyel uygulamalarda önemli iyileştirmeler sağlamıştır.

2024'e gelindiğinde, Adegboye ve arkadaşları **Dual Guided Strategy with Sine Cosine Search Optimization (DGS-SCSO)**'yu, Li ve ekibi ise **Reinforced Golden Sine and Arithmetic Optimization Algorithm (RGAOA)**'yı geliştirerek GSA'nın evrimini sürdürmüştür [17], [18] . Bu yeni algoritmalar, özellikle yüksek boyutlu optimizasyon problemlerinde ve karmaşık mühendislik uygulamalarında dikkat çekici performans artışları elde etmiştir.

Tüm bu çalışmalar, GSA'nın meta-sezgisel optimizasyon alanında güçlü bir temel oluşturduğunu ve farklı algoritmalarla hibrit edildiğinde çeşitli problem türlerinde etkili sonuçlar ürettiğini göstermektedir. GSA'nın basit yapısı ve güçlü matematiksel temeli, onu sürekli geliştirmeye ve yeni uygulamalara adapte etmeye uygun bir algoritma haline getirmektedir.

Kullanım Alanları

- Matematiksel optimizasyon
- Enerji sistemleri
- Endüstriyel mühendislik ve tasarım uygulamaları

İki algoritmanında ilk çalışmalarının Google Scholar üzerinden alınmış ekran görüntüleri Figure 1 ve Figure 2'de verilmiştir.

Figure 1. Bald Eagle Algoritması ilk çalışma.

Figure 2. Golden Sine Algoritması ilk çalışma.

Algoritmaların isimleri ile Web of Science üzerinde yapılan anahtar kelime aramasına göre yapılan çalışma sayılarının grafiği Figure 3 ve Figure 4'de verilmiştir.

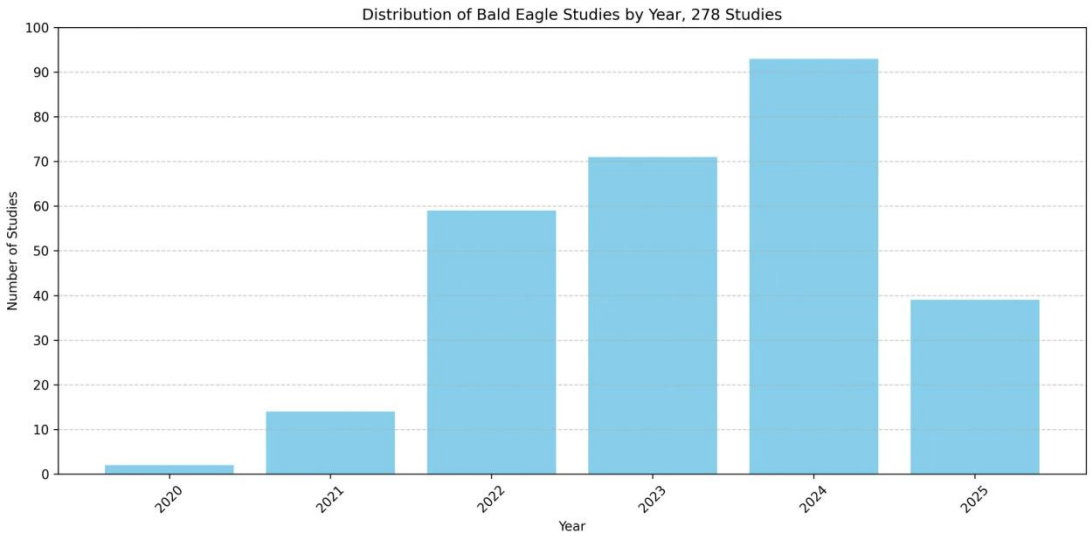


Figure 3. Yıllara göre BES Algoritması üzerine yapılmış akademik çalışma sayıları (Veriler WoS üzerinden alınmıştır).

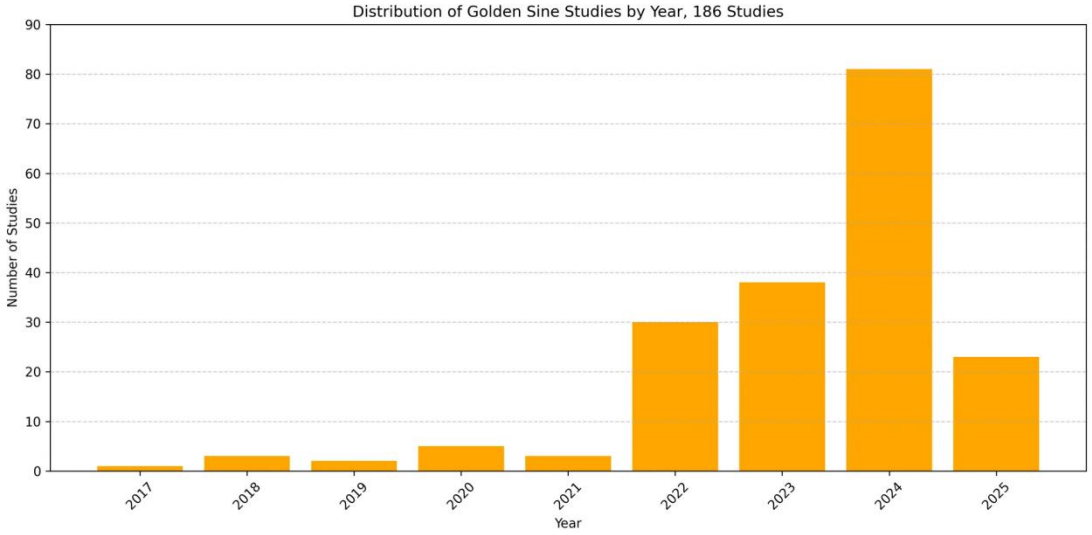


Figure 4. Yıllara göre GSA üzerine yapılmış akademik çalışma sayıları (Veriler WoS üzerinden alınmıştır).

2. Materyal : Algoritmaların Teknik Detayları

Metasezgisel algoritmalar farklı alanlardan ilham almış olabilir. Genel bir sınıflandırma Figure 5'de verilmiştir.

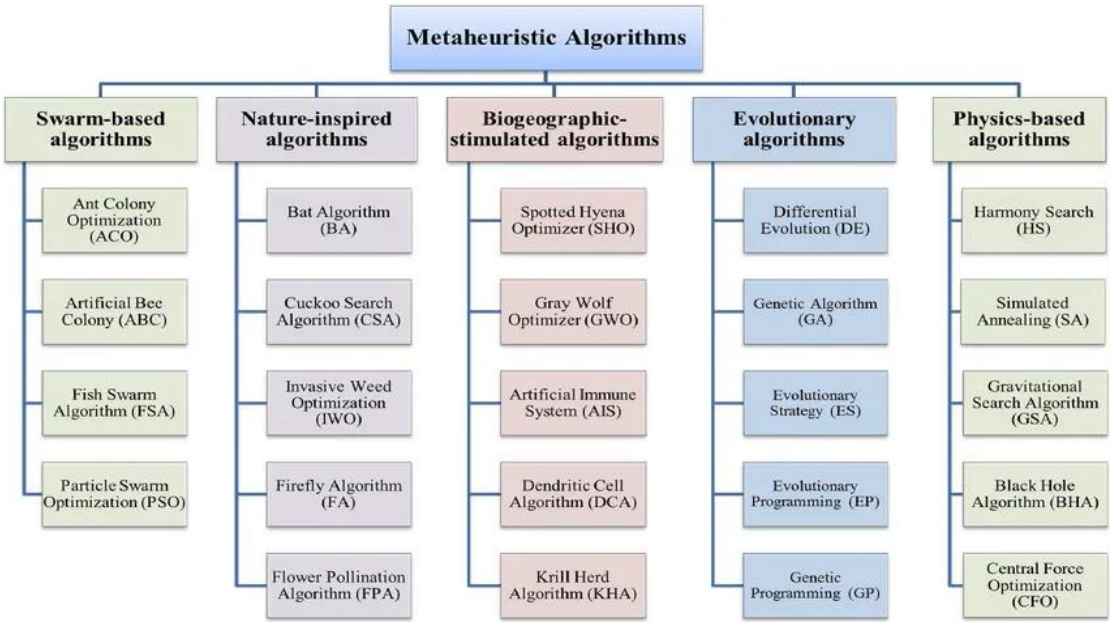


Figure 5. Metasezgisel algoritmaların genel sınıflandırılması

2.1 Bald Eagle Search Algorithm (BES)

Kartallar son derece keskin bir görüşe sahiptir ve havada uzun süre süzülerek geniş bir alanı tarayabilir; bu sayede yüzlerce feet yükseklikteyken bile avlarını tepit edebilirler. Bald Eagle Search (BES), kel kartalların bu avlanma davranışını taklit eden, doğadan ilham alan ve sürü tabanlı bir meta-sezgisel algoritmadır. BES üç temel aşamadan oluşur: **Alan Seçme (Select Space), Alanda Arama (Search in Space) ve Süzülerek Dalış (Swooping)**. İlk olarak, başlangıç popülasyonu (rastgele potansiyel çözümler kümesi) oluşturulur ve değerlendirilir. Sonrasında BES'in her iterasyonu üç alt adım şeklinde ilerler: seçim aşaması ile yeni arama alanları belirlenir, arama aşaması ile bu alanlar içinde çözüm adayları dolaştırılarak en iyi nokta iyileştirilmeye çalışılır, ve dalış aşaması ile bulunan en iyi noktaya doğru tüm adaylar yaklaştırılarak çözüm yoğunlaştırılır. Her aşama, kartalın avlanma davranışının bir parçasını temsil eder ve bu biyolojik analoji sayesinde algoritmanın parametreleri ve adımları tasarlanmıştır.

Adımları

1. **Başlatma**: Rastgele potansiyel çözümlerden oluşan bir başlangıç popülasyonu oluşturulur ve bu çözümler değerlendirilir.
2. **Seçim Aşaması (Exploration)**: Kartalın geniş alanlarda avını araması gibi, algoritma çözüm uzayında yeni arama alanları belirler. Bu aşama, keşfi ve çeşitliliği teşvik eder.

Figure 6. Seçim aşaması - Select Space [1]

3. **Arama Aşaması (Exploitation)**: Seçilen alanlar içinde aday çözümler dolaştırılır ve daha iyi çözümler elde edilmeye çalışılır. Amaç, yüksek kaliteli bölgelere odaklanmaktır.

Figure 7. Arama aşaması - Search Space [1]

4. **Dalış Aşaması (Swooping)**: Bulunan en iyi çözüme doğru tüm adaylar yönlendirilerek çözüm yoğunlaştırılır. Bu, kartalın avına dalışına benzer şekilde gerçekleştirilir.

Figure 8. Dalış aşaması - Swoop [1]

5. **Sonlandırma**: Belirlenen iterasyon sayısına ulaşıldığında veya durma kriteri sağlandığında algoritma sonlandırılır.

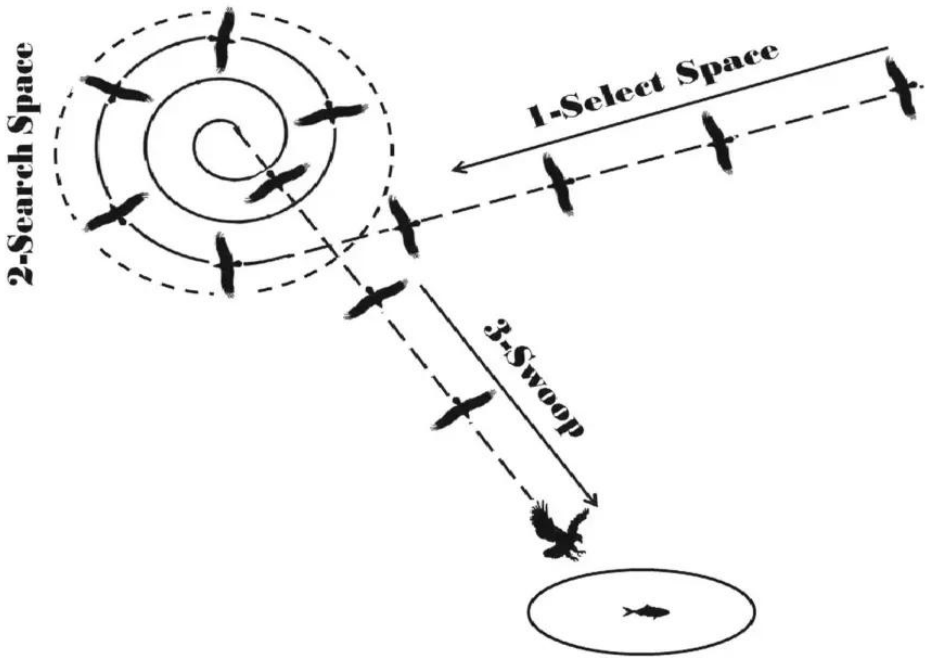


Figure 9. Tüm arama adımlarının resmedilmiş hali (Orjinal makaleden alınımştır) [1]

Aşama 1: Seçim Aşaması (Select Stage)

Bu aşama, kartalların avlanmak için en uygun bölgeyi seçmesini taklit eder. **Kartallar, önceki deneyimlerine (global en iyi pozisyon) ve diğer kartalların konumlarına (ortalama pozisyon) dayanarak en çok potansiyel av içeren bir alanı belirler**. Bu aşama, algoritmanın keşif (exploration) yeteneğini vurgular. *i.* kartalın bu aşamadaki yeni pozisyonu şu şekilde hesaplanır:

$$\mathbf{P}_{\text{new},i}(t+1) = \mathbf{P}_{\text{best}}(t) + a \cdot r \cdot (\mathbf{P}_{\text{mean}}(t) - \mathbf{P}_i(t)) \qquad (1)$$

- $P_{\text{new},i}(t+1)$: i-inci kartalın alan seçme aşaması sonucundaki yeni potansiyel pozisyonudur.
- $P_{\text{best}}(t)$: Mevcut iterasyondaki en iyi kartal pozisyonudur.
- **a** : Pozisyon değişikliklerini kontrol eden bir parametredir ve [1.5, 2] aralığında bir değer alır.

  Not: Bu parametre, Aşama 2 ve 3'teki 'a' parametresinden farklıdır
- r : [0, 1] aralığında rastgele bir sayıdır.
- $P_{\text{mean}}(t)$: Mevcut iterasyondaki tüm kartal pozisyonlarının ortalamasıdır.
- $P_i(t)$: i-inci kartalın mevcut pozisyonudur.

Bu denklem her adayın konumunu, popülasyondaki ortalama ve en iyi birey arasındaki fark doğrultusunda rastgele bir oranda değiştirerek yeni arama bölgesine taşır. Böylece kartal

mevcut en iyi noktaya yakın ancak ondan biraz farklı bir bölgeyi hedefler; bunu yaparken tüm sürünün (popülasyonun) önceki tecrübesini (ortalama noktayı) ve en iyi av bilgisini kullanır.

Aşama 2: Arama Aşaması (Search Stage)

Arama aşaması, kartalın belirlenen av bölgesi içinde **süzülerek avını arama davranışına** dayanır. Kartal bu aşamada seçtiği alanın içinde daireler çizerek veya spiral bir yol izleyerek farklı yönlere hareket eder; böylece avın tam yerini tespit etmeye çalışır. **Bu sürekli hareket ve tarama, kartalın hedefini netleştirmesini ve dalış için en iyi pozisyonu almasını sağlar.** Bu aşama, sömürü (exploitation) yeteneğini artırır. **Bu aşamanın matematiksel arkaplanında, koordinat düzleminde polar (kutupsal) koordinatlar kullanarak spiral bir arayış yapmak vardır.** Bu hareket, kutupsal koordinatlar kullanılarak Eşitlik (2)'deki gibi modellenir:

$$P_{i,new}(t+1) = P_i(t) + y(i) * (P_i(t) - P_{i+1}(t)) + x(i) * (P_i(t) - P_{mean}(t)) \quad (2)$$

Burada $P_{i+1}(t)$ bir sonraki kartalın pozisyonudur. $(P_i - P_{i+1})$ terimi, komşu iki aday konumu arasındaki farkı temsil ederken, $(P_i - P_{mean})$ terimi, adayın popülasyonun merkeziyle olan konumsal farkını ifade eder. Bu farklar sırasıyla $y(i)$ ve $x(i)$ ile çarpılarak mevcut konuma eklenir. $y(i)$ ve $x(i)$ değerleri Eşitlik 3'de verildiği gibi hesaplanır.

$$x(i) = \frac{\mathbf{x}_r(i)}{\max(|\mathbf{x}_r|)}, \quad y(i) = \frac{\mathbf{y}_r(i)}{\max(|\mathbf{y}_r|)} \qquad (3)$$

- $\boldsymbol{x(i)}, \boldsymbol{y(i)}$: her bir aday için rastgele üretilen spiral yönelim katsayılarıdır. Elde edilen sayı max değere bölünerek normalleştirilir ve [-1, 1] aralığına getirilir.
- $x_r(i)$, $y_r(i)$: koordinat değerleridir. Bu değerler Eşitlik 4'te verilen denklemler ile hesaplanır.

$$x_r(i) = r(i) * sin(\theta(i))$$
$$y_r(i) = r(i) * cos(\theta(i)) \qquad (4)$$

- $r(i)$: spiral hareketin yarıçapını belirler. Bu değer Eşitlik 5'de verilen denklemler ile hesaplanır.

$$r(i) = \theta(i) + R * rand()$$
$$\theta(i) = a * \pi * rand() \qquad (5)$$

- $\theta(i)$: $i.$ aday için rastgele bir açısal değerdir. a katsayısı ile ölçeklendirilmiştir. (0π ile 10π arasında bir değer alır yani kartalın 2-5 tam tur arasında rastgele bir dönüş yapmasını sağlar)

- a : kartalın spiralinin açıklığını (dönüş açısının büyüklüğünü) kontrol eder ve genellikle 5 ile 10 arasında seçilir
- R : 0.5 ile 2 arasında değer alan ve yarıçapa eklenen pay.

Aşama 3: Dalış Aşaması (Swooping Stage)

Kartal, önceki aşamada belirlediği en iyi konumdan (hedefin hemen üstünden) **süzülerek dalışa geçer** ve yüksek hızla avını yakalamaya çalışır. Tüm dikkatini ve hareketlerini o hedefe odaklar, diğer kartallar da (eğer birlikte avlanma durumu varsa) en iyi av noktasına yönelebilir. Bu davranış, artık keşiften ziyade yakalamaya, yani hedefe kilitlenmeye odaklıdır. Bu hareket algoritma içerisinde Swooping stage olarak adlandırılır. Arama aşamasında bulunmuş olan P_{best} (o iterasyondaki en iyi aday), bu aşamada merkezi bir rol oynar. Kartal davranışına benzer şekilde, P_{best} noktasına doğru "süzülme" matematiksel olarak Eşitlik 6 ile ifade edilir.

$$\mathbf{P}_{i,\text{new}}(t+1) = \text{rand}() \cdot \mathbf{P}_{\text{best}}(t) + x_1(i) \cdot (\mathbf{P}_i(t) - c_1 \cdot \mathbf{P}_{\text{mean}}(t)) + y_1(i) \cdot (\mathbf{P}_i(t) - c_2 \cdot \mathbf{P}_{\text{best}}(t)) \quad (6)$$

$rand(\ )$ fonksiyonu ile [0, 1] arasında rastgele bir sayı alınarak P_{best} ile çarpılır. Bu, çözümlerin birbirinin üzerine çökmesini engeller ve algoritmanın çeşitliliğini artırır. Denklemin ikinci ve üçüncü terimleri ise sırasıyla adayın ortalama noktadan farkı ve en iyi noktadan farkı üzerinden bir düzeltme ekler. c_1, c_2 katsayıları bu iki farkın ne ölçüde hesaba katılacağını belirler ve tipik olarak 1 ile 2 arasında seçilir (makalede c_1 = c_2 = 2 alınmakta). $x_1(i)$, $y_1(i)$ değerleri ise arama aşamasında olduğu gibi polar koordinatı ifade eden ve [-1, 1] arasında normalleştirilen değerlerdir. Bu aşamada kartalın hareketi doğrusal olmaktan çok hiperbolik bir eğriyi andırdığından, trigonometrik fonksiyonlar yerine hiperbolik fonksiyonlar kullanılmıştır. İşlemler Eşitlik 7, 8 ve 9'da verilmiştir ve önceki adımlar ile benzerdir.

$$x_1(i) = \frac{\mathbf{x}_r(i)}{\max(|\mathbf{x}_r|)}, \quad y_1(i) = \frac{\mathbf{y}_r(i)}{\max(|\mathbf{y}_r|)} \quad (7)$$

$$\begin{aligned} x_r(i) &= r(i) * sinh(\theta(i)) \\ y_r(i) &= r(i) * cosh(\theta(i)) \end{aligned} \quad (8)$$

$$\begin{aligned} r(i) &= \theta(i) \\ \theta(i) &= a * \pi * rand() \end{aligned} \quad (9)$$

Önceki aşamadan farklı olarak burada $r(i)$ ile $\theta(i)$ arasında sabit bir ilişki vardır. Bu arama aşamasındaki spiral yerine dalışta daha doğrusal bir atılım sağlanması gerektiğindendir. Özellikle hiperbolik sin ve cos fonksiyonlarının seçimi ile son aşamalarda yüksek değerler elde edilebilir. Bu da dalışın son kısmındaki hızlı yaklaşmayı modellemektedir. Sonuç olarak, dalış aşaması güncelleme denklemi her bir adayın konumunu P_{best} 'e yaklaştırırken P_{mean}'den uzaklaştıracak şekilde dengelemiştir.

BES Algoritması Sözdekodu ve Akış Diyagramı

```
1:   Randomly initialise Point Pi for n_Point;
2:   Calculate the fitness values of initial Point:
f(Pi);
3:   while (the termination conditions are not met)
4:       // Select space
5:       for (each point i in the population)
6:           P_new = P_best + α * rand(P_mean − P_i)
7:           if f(P_new) < f(P_i)
8:               P_i = P_new
9:               if f(P_new) < f(P_best)
10:                  P_best = P_new
11:              end if
12:          end if
13:      end for
14:      // Search in space
15:      for (each point i in the population)
16:          P_new = P_i + y(i) * (P_i − P_{i+1}) +
x(i) * (P_i − P_mean)
17:          if f(P_new) < f(P_i)
18:              P_i = P_new
19:              if f(P_new) < f(P_best)
20:                  P_best = P_new
21:              end if
22:          end if
23:      end for
24:      // Swoop
25:      for (each point i in the population)
26:          P_new = rand * P_best + x1(i) * (P_i − c1
* P_mean) + y1(i) * (P_i − c2 * P_best)
27:          if f(P_new) < f(P_i)
28:              P_i = P_new
29:              if f(P_new) < f(P_best)
30:                  P_best = P_new
31:              end if
32:          end if
33:      end for
34:      Set k := k + 1;
35:  end while
36:  return P_best
```

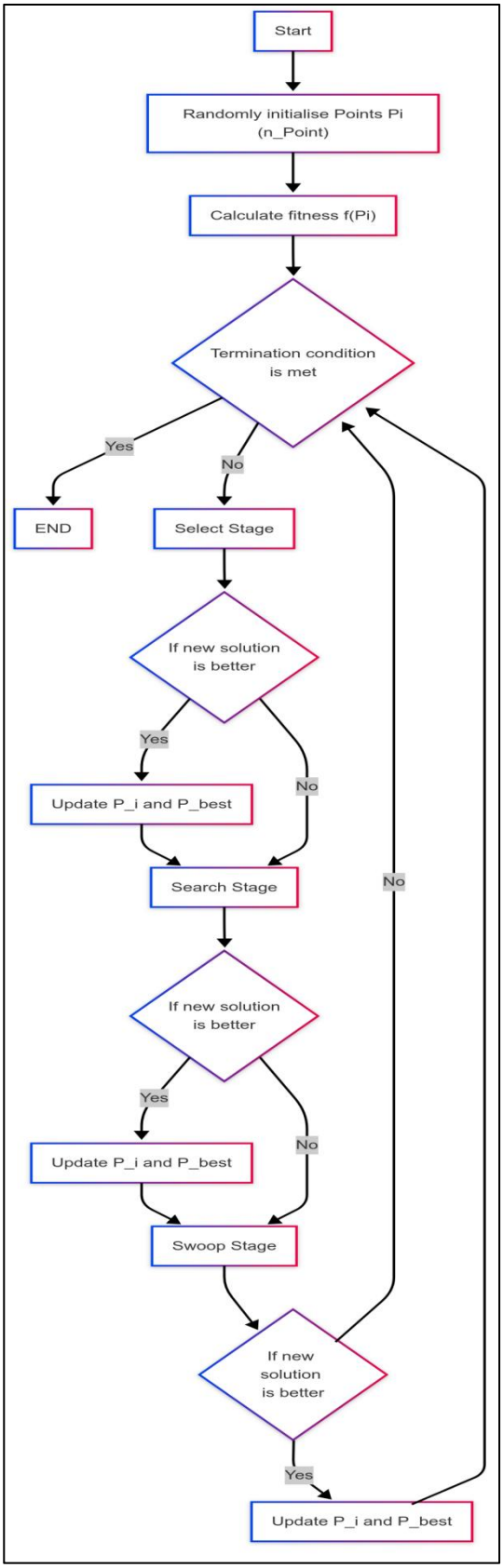


Figure 10. BES algoritması akış diyagramı

2.2 Golden Sine Algorithm (Gold-SA)

Golden Sine algoritması popülasyon tabanlı bir meta-sezgisel optimizasyon algoritmasıdır. **Sinüs fonksiyonundan** ve **altın oran** prensibinden ilham alır. Bu sayede arama sürecinde hem rastgelelik ve çeşitlilik (sinüs dalgalanması) sağlanmakta, hem de en iyi çözüme doğru kademeli bir yakınsama gerçekleştirilmektedir.

Algoritmanın temel yapısı şöyledir: Başlangıçta çözüm uzayında geniş bir alana yayılmış rasgele çözümler üretilir. Her iterasyonda, mevcut en iyi çözüm (hedef çözüm) belirlenir ve diğer aday çözümler bu hedefe doğru **sinüs fonksiyonlu bir güncelleme operatörü** ile hareket ettirilir. Güncelleme operatörü, sinüs fonksiyonunun periyodik ve dalgalı yapısını kullanarak çözümlere **dengeleyici bir arama hareketi** kazandırır. Sinüs fonksiyonu 2π periyotlu olup [−1, 1] aralığında değerler alır; birim çember üzerinde sinüs değerlerinin taranması, aslında çözüm uzayının taranmasına benzer bir süreç olarak görülebilir. Bu benzerlik Gold-SA algoritmasının sinüs tabanlı tasarımına ilham vermiştir. Sinüs fonksiyonunun **döngüsel dalgalanması**, arama işlemini zaman zaman hedef noktadan uzağa taşıyarak yerel minimuma sıkışmayı önlemeye yardımcı olur.

Öte yandan Gold-SA arama sürecinin etkinliğini artırmak için **altın oran** prensibini kullanarak **çözüm alanını daraltır**. Altın oran, matematik ve sanatta uyumlu oranları tanımlayan ünlü bir sayıdır (~1.618) ve **altın kesit arama yöntemi** genellikle maksimum/minimum ararken aralık daraltmak için kullanılır. Gold-SA, her iterasyonda en iyi çözüme odaklanmak üzere arama yapılan aralığı altın orana göre küçültür; böylece tüm arama uzayı yerine, iyi sonuç vermesi muhtemel bölgelere yoğunlaşır. Gold-SA bu avantajlardan yararlanarak arama sürecini verimli hale getirir.

Özetle, başlangıçta geniş bir alanda **keşif (exploration)** yapıp, iterasyonlar ilerledikçe **sömürü (exploitation)** aşamasına geçer. Sinüs fonksiyonunun sağladığı **rastgele periodik hareket** ile keşif davranışı korunurken, altın oran sayesinde arama aralığı giderek daraltılarak en iyi çözüme yakın bölgelerde daha detaylı arama yapılır. Bu yapı, algoritmanın hem geniş alanları taramasını hem de hızlı yakınsama sağlamasını hedefler. Nitekim orijinal çalışmada Gold-SA'nın diğer popülasyon tabanlı yöntemlere kıyasla daha az parametreye sahip olduğu ve daha hızlı yakınsadığı rapor edilmiştir.

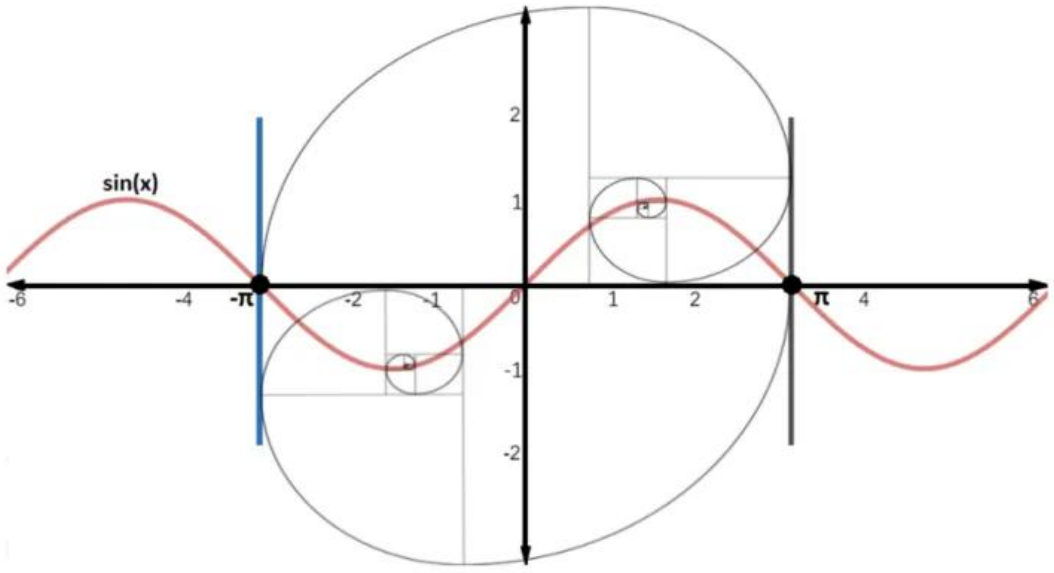


Figure 11. Golden Sine Algoritması (orjinal makaleden alınmıştır.) [10]

Başlangıç Popülasyonu

Gold-SA, **başlangıç popülasyonunu rastgele** oluşturarak arama sürecine başlar. Popülasyon büyüklüğü (ajan sayısı) N ve problem boyutu d olmak üzere, her bir aday çözüm $V_i(i = 1,2, ..., N)$ aşağıdaki şekilde sınırlar içerisinde uniform dağılımlı rastgele değerlerle üretilir. Eşitlik 10'da verilmiştir.

$$V = rand(N, d) \times (\text{upper bound - lower bound}) + \text{lower bound} \quad (10)$$

Burada $rand(N, d)$ ifadesi [0,1] aralığında uniform **rastgele sayılardan oluşan $N \times d$ boyutlu bir matristir**; bu matris, her bir boyut için ilgili aralık (alt sınır, üst sınır) ile ölçeklendirilip kaydırılarak başlangıç çözümlerini verir. Amaç, her boyuttaki değişken için arama alanının tamamına yakınını kaplayacak şekilde çeşitli başlangıç noktaları elde etmektir. Rastgele başlangıç, doğası gereği optimuma uzak noktalar üretebilir; bu da ilk iterasyonlarda algoritmanın düşük kaliteli çözümlerle başlamasına yol açabilir. Eğer popülasyon boyutu çok küçük seçilirse, rastgelelik arama uzayını yeterince kaplayamayabilir ve bazı önemli bölgeler keşfedilmeyebilir. Aşırı büyük popülasyon ise hesaplama maliyetini artırır. Dolayısıyla, başlangıç popülasyonu adımında çeşitlilik ile hesaplama yükü arasında bir denge kurmak gerekir.

Çözüm Güncelleme Adımı (Sinüs Fonksiyonu Kullanımı)

Başlangıç popülasyonu oluşturulduktan sonra, Gold-SA iteratif olarak her bir aday çözümün konumunu güncelleyerek daha iyi çözümler arar. İterasyonun başında popülasyondaki en iyi birey belirlenir ve bu **hedef çözüm (D)** olarak kabul edilir. Ardından, her bir ajan i ve her bir boyut j için $V(i,j)$ değeri **sinüs tabanlı operatör** ile güncellenir. Eşitlik 11'de verilmiştir.

$$V(i,j)' = V(i,j) \times |(sin(r_1)| - r_2 \times sin(r_1) \times |x_1 \times D(j) - x_2 \times V(i,j)| \quad (11)$$

- $V(i,j)$ ifadesi, i. arama ajanının j. boyuttaki mevcut çözüm değeridir.
- $D(j)$, hedef çözüme (popülasyondaki en iyi çözüme) ait j. boyut değeridir. Bu, algoritmanın o andaki **hedef değeri** olarak düşünülebilir.
- r_1 ve r_2 rassal sayılardır: $r_1 \in [0,2\pi]$ ve $r_2 \in [0,\pi]$ aralıklarında uniform rastgele üretilir. Her güncellemede bu rassal değerler yeniden seçilerek operatöre **stokastik bir doğa** kazandırılır.
- x_1 ve x_2 katsayıları, altın oran prensibiyle elde edilen sabitlerdir. Genel olarak bu katsayılar $-\pi$ ile π arasında belirlenen bir aralığın belli oranlı iki noktasını temsil eder ve arama alanını daraltmak için kullanılır.
- $\sin(r_1)$ fonksiyonu, güncellemedeki **sinüsoidal hareketin** temelidir. $|\sin(r_1)|$ terimi mevcut çözüm değerini ölçeklendirmekte, $\sin(r_1)$ teriminin kendisi ise ikinci kısımdaki fark değerine etki etmektedir.

Güncelleme formülünün ilk kısmı $V(i,j) \times |\sin(r_1)|$, mevcut çözüm değerini 0 ile 1 arasında bir orana indirger. Çözümün değerini belirli bir oranda küçülterek değişim adımının kontrollü olmasını sağlar. İkinci kısım ise hedef değer $D(j)$ ile mevcut değer $V(i,j)$ arasındaki farkın mutlak değerini alır ve bunu $\sin(r_1)$ile çarparak bir **dalgalanma unsuru** ekler. Bu fark, r_2 ile ölçeklendirilip mevcut değerden çıkarılır. Sonuç olarak, $D(j)$ ve $V(i,j)$ terimlerine bağlı olarak hedefe yaklaşılır veya geçici olarak hedefin ötesine geçilir. **Sinüs fonksiyonunun pozitif veya negatif olabilmesi**, güncelleme adımına bu şekilde bir ileri-geri salınım özelliği kazandırır. Bu sayede algoritma her adımda doğrudan en iyi çözüme yapışıp kalmaz; onun yerine etrafındaki alanda titreşimler yaparak daha geniş bir aramaya imkan tanır.

- Gold-SA'nın yenilikçi yönü, bu rastgeleliği **matematiksel bir dalga** olan sinüs fonksiyonu ile sağlaması ve bunu **altın oran katsayılarıyla** birleştirerek hedefe yaklaşmayı dinamik bir şekilde kontrol etmesidir.

Her iterasyon sonunda popülasyondaki tüm çözümlerin uygunluk değerleri (objective value) yeniden hesaplanır ve yeni bir en iyi çözüm bulunursa **hedef D güncellenir**. Gold-SA bu döngüyü belirlenen maksimum iterasyon sayısına ulaşana veya tatmin edici bir çözüm elde edilene kadar sürdürür. İterasyonlar ilerledikçe, altın oran mekanizması nedeniyle arama alanı daraldığı için adımlar daha küçük ve ince ayarlı hale gelir; sinüs tabanlı hareket ise son ana dek çeşitlilik sağlayarak algoritmanın sıkışmasını engeller.

Altın Oran ve Arama Alanının Daraltılması

Algoritma'nın kendine özgü yönlerinden biri, **altın oran prensibini kullanarak arama aralığını daraltması** ve böylece optimizasyon sürecini hızlandırmasıdır. **Altın kesit arama yöntemi**, bir fonksiyonun maksimum veya minimumunu bulmak için aralık daraltmaya yarayan klasik bir yöntemdir. Bu yöntemde, a, b aralığında seçilen iki nokta aracılığıyla (genellikle orantılar altın oranla belirlenir) fonksiyon değerlendirmeleri yapılır ve her adımda aralık en verimli şekilde küçültülür. Altın oranın matematiksel tanımı şöyledir: eğer a ve b pozitif sayılarına sahip bir doğru parçası altın orana göre bölünmüşse, büyük parçanın küçük parçaya oranı, bütünün büyük parçaya oranına eşittir.

$$\frac{a+b}{a} = \frac{a}{b} = \varphi \tag{12}$$

Yukarıdaki ilişkiyi sağlayan φ sayısı **altın oran** olarak adlandırılır. Bu denklemi çözerek φ'nin değeri aşağıdaki gibi bulunur:

$$\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618033\ldots \tag{13}$$

(Altın oran ifadesinin tersinin, $\frac{\sqrt{5}-1}{2} \approx 0.618033$ değerine eşit olduğu ve aralık daraltmada bu oranın da kullanıldığı not edilmelidir).
Altın kesit arama yönteminde, bir aralık içerisindeki optimum noktayı bulmak için her iterasyonda aralık belirli bir oranda küçültülür. Örneğin a , b aralığında, altın oran kullanılarak iç nokta seçimleri Eşitlik 14'de verldiği gibi yapılır.

$$x_1 = a + (1-\varphi)(b-a), \quad x_2 = a + \varphi(b-a) \tag{14}$$

Bu seçimler sonrasında $f(x_1)$ ve $f(x_2)$ değerleri karşılaştırılır; uygun aralığın yarısı elenir ve kalan aralık yine altın oran oranında bölünerek süreç tekrarlanır. Bu yöntemin avantajı, her adımda yalnızca tek bir yeni nokta değerlendirmeye ihtiyaç duyulması (diğer nokta önceki adımda hesaplanmıştır) ve aralığın sabit bir oranda (φ) küçültülmesidir. Böylece oldukça hızlı yakınsama sağlanır ve türev gibi ekstra bilgilere ihtiyaç duyulmaz.

- Gold-SA'nın altın kesit kullanımına ilişkin orijinal çalışmada, özel durumlar için bazı ek tedbirler de belirtilmiştir. Örneğin, x_1 ve x_2 değerlerinin birbirine eşit çıkması istenmez; bu durum, altın kesit formülünde çok nadir de olsa sayısal yuvarlamalarla meydana gelebilir. Böyle bir durumda algoritma, x_1 ve x_2 'yi yeniden rastgele (örneğin $0, \pi$ aralığında farklı) değerlere çekip formül (14) ile **yeniden hesaplar.** Bu sayede güncelleme formülünde payda sıfır olması gibi istenmeyen durumlardan kaçınılır. Altın oran katsayılarının başarılı bir şekilde uygulanmasıyla, Gold-SA sözde kodunda her iterasyon sonunda arama aralığının daralması sağlanmış olur (orijinal makalede algoritmanın tam sözde kodu bu adımlarla verilmiştir).

Gold-SA Sözdekodu ve Akış Diyagram

```
1.  Randomly generate the initial population V with Equation 2
2.  Calculate the fitness f(V(i , *)) of all search agents
3.  Find the best search agent and assign its vector to D(j)  ← target value
4.  while maximum number of iterations
5.      for i = 1 to N    do
6.          r  ← rand(0,1)
7.          r1 ← 2π * r
8.          r2 ← π  * r
9.          for j = 1 to D    do
10.             V(i , j) ← V(i , j) * |sin(r1)| − r2 * sin(r1) * | x1 * D(j) −
 x2 * V(i , j) |
11.         end for
12.     end for
13.     Find the best solution in V and assign it to D(j) as the new target value
14.     for j = 1 to D  do
15.         if  V(i* , j) < D(j) then
16.             b ← x2 ;    x2 ← x1
17.             x1 ← a * τ + b * (1 − τ)
18.         else
19.             a ← x1 ;    x1 ← x2
20.             x2 ← a * (1 − τ) + b * τ
21.         end if
22.         if  x1 == x2    then
23.             a, b ← rand(0, π), rand(0, −π)
24.             x1 ← a * τ + b * (1 − τ)
25.             x2 ← a * (1 − τ) + b * τ
26.         end if
27.     end for
28.     t ← t + 1
29. end while
30. return  {V} , best = f(D)
```

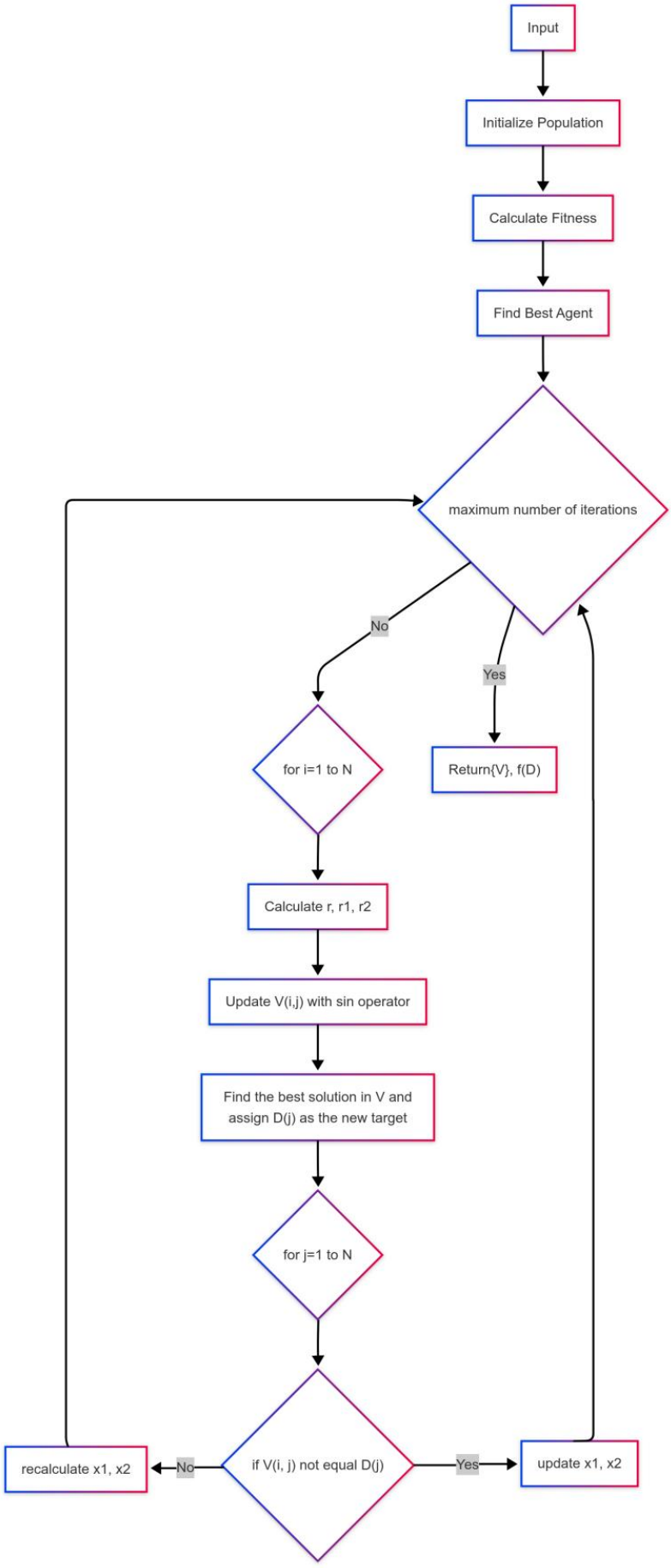

Input
Initialize Population
Calculate Fitness
Find Best Agent
maximum number of iterations
No
Yes
for i=1 to N
Return{V}, f(D)
Calculate r, r1, r2
Update V(i,j) with sin operator
Find the best solution in V and assign D(j) as the new target
for j=1 to N
if V(i, j) not equal D(j)
No
Yes
recalculate x1, x2
update x1, x2

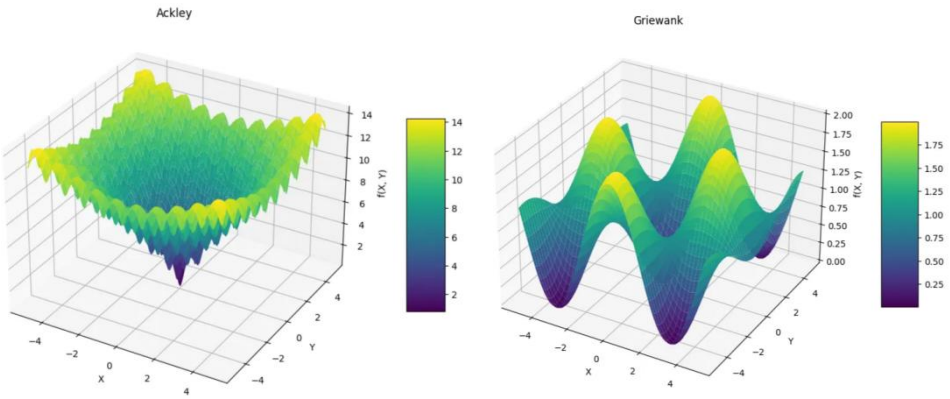
Figure 13. Gold-SA akış diyagramı

3. Algoritmaların Test Sonuçları

Bu bölümde, her iki algoritma literatürde kullanılan test fonksiyonlarına uygulanacaktır. bu fonksiyonlar Figure 14'de verilmiştir.

#	Fonksiyon (n boyut)	Matematiksel denklem	Tipik global minimum
1	Sphere	$f(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$	$f = 0 \ @\mathbf{x} = 0$
2	Rosenbrock	$f(\mathbf{x}) = \sum_{i=1}^{n-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right]$	$f = 0 \ @\mathbf{x} = (1, \ldots, 1)$
3	Ackley	$f(\mathbf{x}) = -a\exp\left(-b\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(c\,x_i)\right) + a + \mathrm{e}$ Varsayılan sabitler $a = 20,\ b = 0.2,\ c = 2\pi$	$f = 0 \ @\mathbf{x} = 0$
4	Rastrigin	$f(\mathbf{x}) = 10\,n + \sum_{i=1}^{n}\left[x_i^2 - 10\cos\left(2\pi x_i\right)\right]$	$f = 0 \ @\mathbf{x} = 0$
5	Griewank	$f(\mathbf{x}) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)$	$f = 0 \ @\mathbf{x} = 0$

Figure 14. Test fonksiyonları



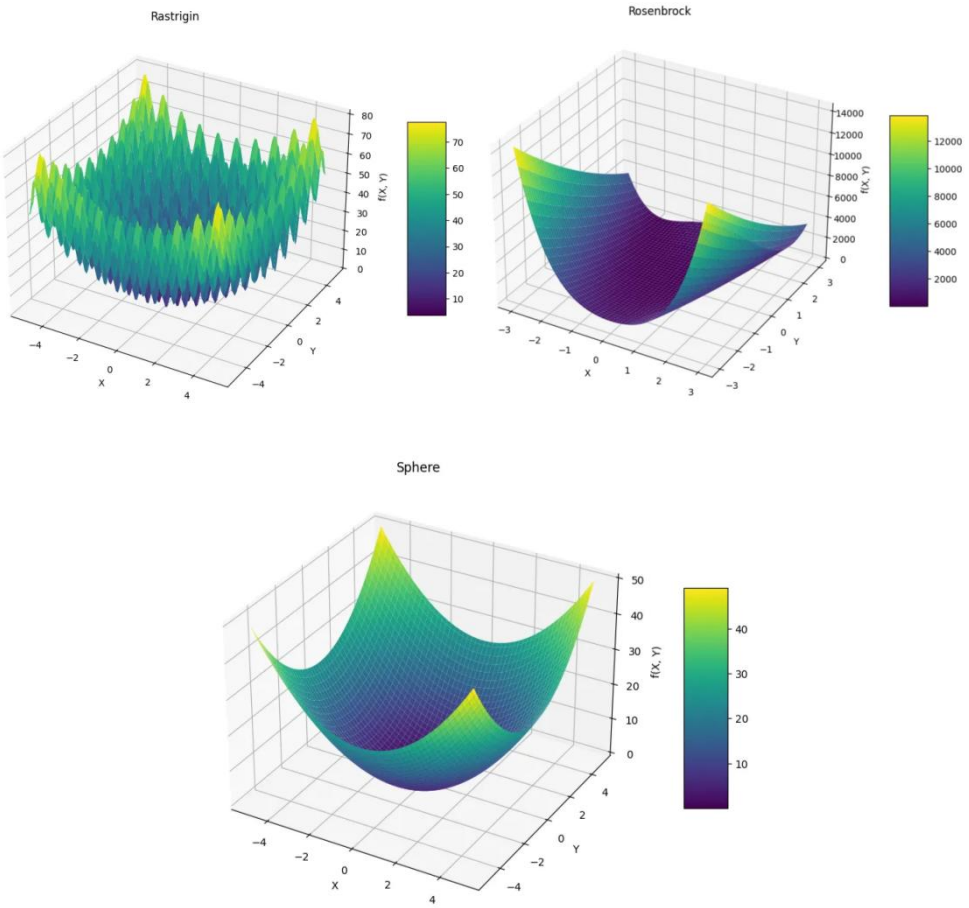


Figure 15. Test fonksiyonları

Test fonksiyonları aynı iterasyon sayısında denenmesi sonucu elde edilen sonuçlar Figure 16'daki gibi olmuştur.

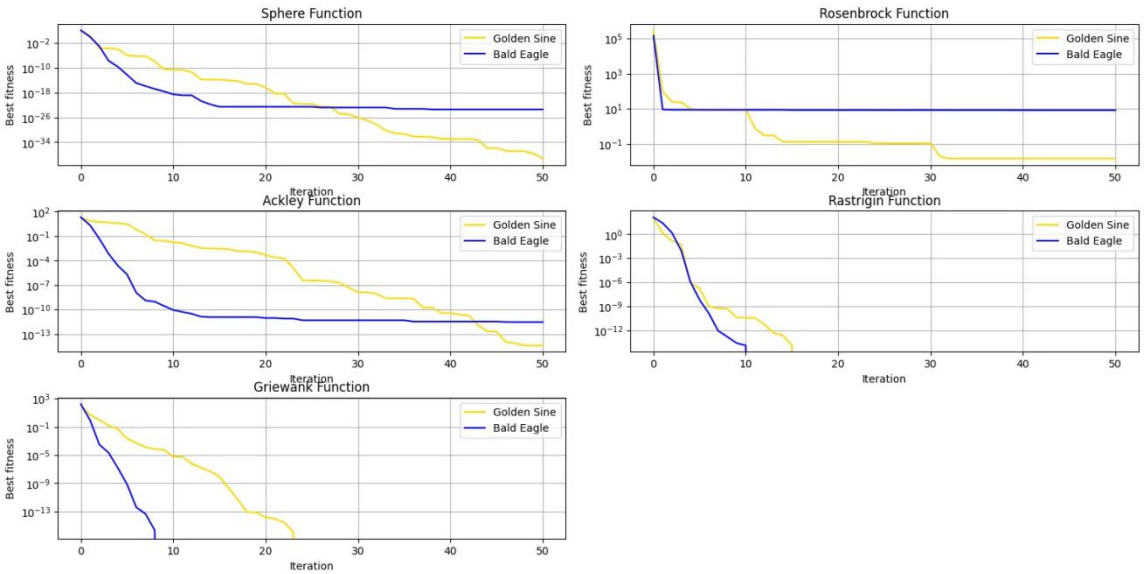


Figure 16. Test fonksiyonu sonuçları

4. Sonuç

Bald Eagle Search algoritması, yukarıda detaylandırılan üç aşamalı yaklaşımı sayesinde optimizasyon problemlerinde hem global arama yapabilme hem de hızlı yakınsama sağlayabilme potansiyeline sahiptir. Kartalların avlanma stratejisinden ilham alan bu yöntem, seçim aşamasında **geniş bir keşif**, arama aşamasında **yönlendirilmiş bir tarama**, ve dalış aşamasında **odaklanmış bir iyileştirme** adımı sunmaktadır. Her bir aşama, belirli parametreler ve denklemlerle kontrol edilmekte olup, algoritmanın davranışı bu sayede esnekçe ayarlanabilir. Alsattar ve arkadaşlarının 2019 tarihli çalışmalarında gösterildiği üzere, BES algoritması pek çok benchmark optimizasyon probleminde gelişmiş meta-sezgisel algoritmalarla rekabet edebilen sonuçlar elde etmiştir. Bu başarı, büyük ölçüde BES'in üç aşamasının dengeli şekilde keşif ve sömürü yapabilmesinden kaynaklanmaktadır.

Sonuç olarak, BES meta-sezgisel algoritması yüksek lisans düzeyinde çalışmalarda dikkat çeken yenilikçi bir yöntemdir. Teknik detayları anlaşılmak suretiyle, farklı problem kümelerine uyarlanabilir ve parametre seçimleriyle performansı iyileştirilebilir.

Altın Sinüs Algoritması, **sinüs dalgalanması ile altın kesit daraltmasını bir araya getiren** yenilikçi bir optimizasyon algoritmasıdır. Matematiksel temellere dayanan bu yaklaşım, az sayıda parametre ile karmaşık problemlerde rekabetçi performans göstermektedir. Gold-SA, özellikle mühendislik optimizasyonu ve benzeri alanlarda diğer meta-sezgisel yöntemlerle karşılaştırıldığında hızlı yakınsama ve yüksek doğruluk potansiyeli sunar. Bununla birlikte, her meta-sezgisel algoritma gibi problemden bağımsız bir sihirli değnek değildir; doğru ayarlar ve gerektiğinde modifikasyonlarla kullanıldığında en iyi sonucu verecektir. . Gold-SA'nın matematiksel sadeliği ve etkinliği, gelecekteki çalışmalar için de ilham kaynağı olmaya devam etmektedir.

5. Referanslar

[1] H. A. Alsattar, A. A. Zaidan, and B. B. Zaidan, "Novel meta-heuristic bald eagle search optimisation algorithm," *Artif Intell Rev*, vol. 53, no. 3, pp. 2237–2264, Mar. 2020, doi: 10.1007/s10462-019-09732-5.

[2] A. Fathy, H. Rezk, D. Yousri, T. Kandil, and A. G. Abo-Khalil, "Real-time bald eagle search approach for tracking the maximum generated power of wind energy conversion system," *Energy*, vol. 249, Jun. 2022, doi: 10.1016/j.energy.2022.123661.

[3] W. Liu *et al.*, "A Hybrid Bald Eagle Search Algorithm for Time Difference of Arrival Localization," *Applied Sciences (Switzerland)*, vol. 12, no. 10, May 2022, doi: 10.3390/app12105221.

[4] S. R. Sharma, M. Kaur, and B. Singh, "A Self-adaptive Bald Eagle Search optimization algorithm with dynamic opposition-based learning for global optimization problems," *Expert Syst*, vol. 40, no. 2, Feb. 2023, doi: 10.1111/exsy.13170.

[5] Y. Zhang, Y. Zhou, G. Zhou, and Q. Luo, "An effective multi-objective bald eagle search algorithm for solving engineering design problems," *Appl Soft Comput*, vol. 145, p. 110585, 2023, doi: 10.24433/CO.991.

[6] S. Kankılıç and E. Karpat, "Optimization of Multilayer Absorbers Using the Bald Eagle Optimization Algorithm," *Applied Sciences (Switzerland)*, vol. 13, no. 18, Sep. 2023, doi: 10.3390/app131810301.
[7] F. S. Gharehchopogh, "An Improved Boosting Bald Eagle Search Algorithm with Improved African Vultures Optimization Algorithm for Data Clustering," *Annals of Data Science*, Apr. 2024, doi: 10.1007/s40745-024-00525-4.
[8] H. Youssef, S. Kamel, M. H. Hassan, L. Nasrat, and F. Jurado, "An improved bald eagle search optimization algorithm for optimal home energy management systems," *Soft comput*, vol. 28, no. 2, pp. 1367–1390, Jan. 2024, doi: 10.1007/s00500-023-08328-0.
[9] M. A. El-Shorbagy, A. Bouaouda, H. A. Nabwey, L. Abualigah, and F. A. Hashim, "Bald eagle search algorithm: a comprehensive review with its variants and applications," 2024, *Taylor and Francis Ltd.* doi: 10.1080/21642583.2024.2385310.
[10] E. Tanyildizi and G. Demir, "Golden sine Algorithm: A novel math-inspired algorithm," *Advances in Electrical and Computer Engineering*, vol. 17, no. 2, pp. 71–78, May 2017, doi: 10.4316/AECE.2017.02010.
[11] E. Tanyildizi, "A novel optimization method for solving constrained and unconstrained problems: Modified Golden Sine Algorithm," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 26, no. 6, pp. 3287–3304, 2018, doi: 10.3906/elk-1802-232.
[12] W. Xie, J. S. Wang, and Y. Tao, "Improved Black Hole Algorithm Based on Golden Sine Operator and Levy Flight Operator," *IEEE Access*, vol. 7, pp. 161459–161486, 2019, doi: 10.1109/ACCESS.2019.2951716.
[13] J. Zhang and J. S. Wang, "Improved whale optimization algorithm based on nonlinear adaptive weight and golden sine operator," *IEEE Access*, vol. 8, pp. 77013–77048, 2020, doi: 10.1109/ACCESS.2020.2989445.
[14] M. Ghanbari and M. Goldani, "Support Vector Regression Parameters Optimization using Golden Sine Algorithm and its application in stock market," Mar. 2021, [Online]. Available: http://arxiv.org/abs/2103.11459
[15] M. Han, Z. Du, H. Zhu, Y. Li, Q. Yuan, and H. Zhu, "Golden-Sine dynamic marine predator algorithm for addressing engineering design optimization," *Expert Syst Appl*, vol. 210, Dec. 2022, doi: 10.1016/j.eswa.2022.118460.
[16] P. Yuan, T. Zhang, L. Yao, Y. Lu, and W. Zhuang, "A Hybrid Golden Jackal Optimization and Golden Sine Algorithm with Dynamic Lens-Imaging Learning for Global Optimization Problems," *Applied Sciences (Switzerland)*, vol. 12, no. 19, Oct. 2022, doi: 10.3390/app12199709.
[17] O. R. Adegboye, A. K. Feda, O. R. Ojekemi, E. B. Agyekum, B. Khan, and S. Kamel, "DGS-SCSO: Enhancing Sand Cat Swarm Optimization with Dynamic Pinhole Imaging and Golden Sine Algorithm for improved numerical optimization performance," *Sci Rep*, vol. 14, no. 1, Dec. 2024, doi: 10.1038/s41598-023-50910-x.
[18] M. Li, Z. Liu, and H. Song, "An improved algorithm optimization algorithm based on RungeKutta and golden sine strategy," *Expert Syst Appl*, vol. 247, Aug. 2024, doi: 10.1016/j.eswa.2024.123262.

6. Ek

Raporu web üzerinde okumak için aşağıdaki link kullanılabilir.
Yasin Ünal - Bald Eagle and Golden Sine Algorithms

Python Kodu :

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from matplotlib import cm
import time

# ------------------------------------------------------------
#  Test fonksiyonları
# ------------------------------------------------------------
class TestFunctions:
    @staticmethod
    def sphere(x: np.ndarray):
        return np.sum(x ** 2)

    @staticmethod
    def rosenbrock(x: np.ndarray):
        return np.sum(100.0 * (x[1:] - x[:-1] ** 2) ** 2 + (x[:-1] - 1) ** 2)

    @staticmethod
    def ackley(x: np.ndarray):
        a, b, c = 20, 0.2, 2 * np.pi
        term1 = -a * np.exp(-b * np.sqrt(np.sum(x ** 2) / len(x)))
        term2 = -np.exp(np.sum(np.cos(c * x)) / len(x))
        return term1 + term2 + a + np.exp(1)

    @staticmethod
    def rastrigin(x: np.ndarray):
        n = len(x)
        return 10 * n + np.sum(x ** 2 - 10 * np.cos(2 * np.pi * x))

    @staticmethod
    def griewank(x: np.ndarray):
        sum_sq = np.sum(x ** 2 / 4000)
        prod_cos = np.prod(np.cos(x / np.sqrt(np.arange(1, len(x) + 1))))
        return 1 + sum_sq - prod_cos


# ------------------------------------------------------------
#  Golden Sine Algorithm
# ------------------------------------------------------------
class GoldenSineAlgorithm:
    def __init__(self, obj_func, n_dim, lb, ub,
                 population_size=30, max_iter=100):
        self.f = obj_func
        self.D = n_dim
        self.lb = np.array([lb] * n_dim) if np.isscalar(lb) else lb.astype(float)
        self.ub = np.array([ub] * n_dim) if np.isscalar(ub) else ub.astype(float)
        self.N = population_size
        self.T = max_iter
        self.tau = (np.sqrt(5) - 1) / 2.0  # golden ratio

    # ------------------------------------------------
    def _clip(self, P):
        return np.clip(P, self.lb, self.ub)

    # ------------------------------------------------
    def optimize(self):
        # init pop
        P = np.random.uniform(self.lb, self.ub, (self.N, self.D))
```

```
        fit = np.array([self.f(x) for x in P])
        gbest = P[np.argmin(fit)].copy()
        best_y = fit.min()
        history = [best_y]

        # golden-section scalar interval (for sine step)
        a_int, b_int = -np.pi, np.pi
        x1 = a_int * self.tau + b_int * (1 - self.tau)
        x2 = a_int * (1 - self.tau) + b_int * self.tau

        for _ in range(self.T):
            r1 = 2 * np.pi * np.random.rand(self.N, 1)
            r2 = np.pi * np.random.rand(self.N, 1)
            abs_sin = np.abs(np.sin(r1))
            sin_r1 = np.sin(r1)

            for i in range(self.N):
                P[i] = (P[i] * abs_sin[i] -
                        r2[i] * sin_r1[i] *
                        np.abs(x1 * gbest - x2 * P[i]))
            P = self._clip(P)

            # fitness
            fit = np.array([self.f(x) for x in P])
            if fit.min() < best_y:
                gbest = P[np.argmin(fit)].copy()
                best_y = fit.min()

            # golden-section scalar update (simple)
            if np.random.rand() < 0.5:
                b_int = x2; x2 = x1
                x1 = a_int * self.tau + b_int * (1 - self.tau)
            else:
                a_int = x1; x1 = x2
                x2 = a_int * (1 - self.tau) + b_int * self.tau

            history.append(best_y)
        return gbest, best_y, history


# ------------------------------------------------------------
# Bald Eagle Search Algoirthm
# ------------------------------------------------------------
class BaldEagle:
    def __init__(self, obj_func, n_dim, lb, ub,
                 pop_size=30, max_iter=100,
                 alpha=2.0, a_spiral=8.0, R_spiral=1.0,
                 c1=1.5, c2=1.5):
        self.f = obj_func
        self.D = n_dim
        self.lb = np.array([lb] * n_dim) if np.isscalar(lb) else lb.astype(float)
        self.ub = np.array([ub] * n_dim) if np.isscalar(ub) else ub.astype(float)
        self.N, self.T = pop_size, max_iter
        self.alpha, self.a_spiral, self.R_spiral = alpha, a_spiral, R_spiral
        self.c1, self.c2 = c1, c2

    # -------------------- helpers
    def _clip(self, P):
        return np.clip(P, self.lb, self.ub)
```

```python
    # -------------------- main optimiser
    def optimize(self):
        P = np.random.uniform(self.lb, self.ub, (self.N, self.D))
        fit = np.array([self.f(x) for x in P])
        gbest = P[np.argmin(fit)].copy()
        history = [fit.min()]

        for _ in range(self.T):
            mean = P.mean(axis=0)

            # ---- 1. Select phase
            r = np.random.rand(self.N, 1)
            P = gbest + self.alpha * r * (mean - P)
            P = self._clip(P)

            # ---- 2. Spiral search phase
            theta = self.a_spiral * np.pi * np.random.rand(self.N, 1)
            rad = theta + self.R_spiral * np.random.rand(self.N, 1)
            xr = rad * np.sin(theta)
            yr = rad * np.cos(theta)
            rest = np.random.uniform(-1, 1, (self.N, self.D - 2))
            spiral_move = np.hstack([xr, yr, rest])
            P = mean + spiral_move
            P = self._clip(P)

            # ---- 3. Swoop phase
            xr_s = rad * np.sinh(theta)
            yr_s = rad * np.cosh(theta)
            x1 = xr_s / (np.abs(xr_s).max() + 1e-12)
            y1 = yr_s / (np.abs(yr_s).max() + 1e-12)
            x1 = np.repeat(x1, self.D, axis=1)
            y1 = np.repeat(y1, self.D, axis=1)
            P = (np.random.rand(self.N, 1) * gbest +
                 x1 * (P - self.c1 * mean) +
                 y1 * (P - self.c2 * gbest))
            P = self._clip(P)

            # -- evaluate
            fit = np.array([self.f(x) for x in P])
            if fit.min() < self.f(gbest):
                gbest = P[np.argmin(fit)].copy()
            history.append(self.f(gbest))
        return gbest, history[-1], history


# ------------------------------------------------------------
#  Benchmarklar
# ------------------------------------------------------------

def run_once(alg, name):
    start = time.time();  sol, best, hist = alg.optimize();  elapsed = time.time() - start
    return {"name": name, "best": best, "time": elapsed, "history": hist}


def test_algorithms(func_name, dim, lb, ub, iters=50, pop=30):
    f = getattr(TestFunctions, func_name)
    gsa = GoldenSineAlgorithm(f, dim, lb, ub, pop, iters)
    bea = BaldEagle(f, dim, lb, ub, pop, iters)
    res_gsa = run_once(gsa, "Golden Sine")
    res_bea = run_once(bea, "Bald Eagle")
```

```
    return {"function": func_name, "dimension": dim, "gsa": res_gsa, "bea": res_bea}


def compare_suite():
    suite = [
        ("sphere",     10, -10,   10),
        ("rosenbrock", 10, -5,    10),
        ("ackley",     10, -32.768, 32.768),
        ("rastrigin",  10, -5.12,  5.12),
        ("griewank",   10, -600,  600)
    ]
    return [test_algorithms(*case) for case in suite]


# ------------------------------------------------------------
#  Convergence & result plots
# ------------------------------------------------------------

def plot_convergence(results):
    plt.figure(figsize=(15, 10))
    for i, result in enumerate(results, 1):
        plt.subplot(3, 2, i)
        plt.plot(result['gsa']['history'], label='Golden Sine', color='gold')
        plt.plot(result['bea']['history'], label='Bald Eagle', color='blue')
        plt.title(f"{result['function'].capitalize()} Function")
        plt.xlabel('Iteration')
        plt.ylabel('Best fitness')
        plt.yscale('log')
        plt.grid(True)
        plt.legend()

    plt.tight_layout()
    plt.show()

def plot_3d_surface(func, bounds, title='Function'):
    x = np.linspace(bounds[0], bounds[1], 100)
    y = np.linspace(bounds[0], bounds[1], 100)
    X, Y = np.meshgrid(x, y)

    Z = np.zeros_like(X)
    for i in range(X.shape[0]):
        for j in range(X.shape[1]):
            Z[i,j] = func(np.array([X[i,j], Y[i,j]]))

    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')

    surf = ax.plot_surface(X, Y, Z, cmap=cm.viridis,
                           linewidth=0, antialiased=True)

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('f(X, Y)')
    ax.set_title(title)

    fig.colorbar(surf, shrink=0.5, aspect=5)
    plt.show()


# ------------------------------------------------------------
```

```python
#  Main
# ------------------------------------------------------------
if __name__ == "__main__":
    print("Running comparison suite…")

    plot_3d_surface(TestFunctions.rosenbrock, (-3,3), title='Rosenbrock')
    plot_3d_surface(TestFunctions.sphere, (-5,5), title='Sphere')
    plot_3d_surface(TestFunctions.ackley, (-5,5), title='Ackley')
    plot_3d_surface(TestFunctions.rastrigin, (-5,5), title='Rastrigin')
    plot_3d_surface(TestFunctions.griewank, (-5,5), title='Griewank')
    results = compare_suite()
    tbl = []
    for r in results:
        tbl.append({
            "Func": r["function"],
            "Best GSA": f"{r['gsa']['best']:.3e}",
            "Best BES": f"{r['bea']['best']:.3e}",
            "t_GSA(s)": f"{r['gsa']['time']:.2f}",
            "t_BES(s)": f"{r['bea']['time']:.2f}"
        })
    print(pd.DataFrame(tbl).to_string(index=False))

    # plot convergence
    plot_convergence(results)
```