INTERNATIONAL SCIENTIFIC AND VOCATIONAL JOURNAL (ISVOS JOURNAL)

Vol.: 9 Issue: 1 Date: 30.06.2025 Received: 27.03.2025 Accepted: 13.05.2025 Final Version: 27.06.2025

ISVOS Journal, 2025, 9(1): 94-117 – DOI: 10.47897/bilmes.1666766

Crayfish Optimization Algorithm

Osman Karataş [a], Celal Yaşar [b,1], Hasan Temurtaş [c], Serdar Özyön [d]

[a] Kütahya Dumlupınar University, Institute of Graduate Education, Kütahya, Türkiye
ORCID ID: 0009-0006-3067-1157
[b] Kütahya Health Science University, Faculty of Engineering and Natural Sciences, Kütahya, Türkiye
ORCID ID: 0000-0002-5069-8545
[c] Kütahya Dumlupınar University, Faculty of Engineering, Kütahya, Türkiye
ORCID ID: 0000-0001-6738-3024
[d] Kütahya Dumlupınar University, Faculty of Engineering, Kütahya, Türkiye
ORCID ID: 0000-0002-4469-3908

Abstract

This study aims to improve the performance of the Crayfish Optimization Algorithm (COA), a swarm intelligence algorithm recently introduced in the literature, on various test functions with fixed and variable dimensions. Optimization can be defined as making a system as efficient as possible at the least cost, within certain constraints. Numerous optimization algorithms have been designed in the literature to obtain the best solutions for specific problems. The most critical aspects in solving these problems are modeling the problem correctly, determining the parameters and constraints, and selecting an appropriate meta-heuristic algorithm for solving the objective function. Not every algorithm is suitable for every problem structure. While some algorithms solve fixed-dimension test functions better, others may perform better on variable-dimension test functions. In this study, the COA algorithm's performance was evaluated on 10 test functions previously used in the literature, consisting of three fixed-dimension functions (Schaffer Function, Himmelblau Function, Kowalik Function) and seven variable-dimension functions, including one unimodal (Elliptic Function) and six multimodal functions (Non-Continuous Rastrigin Function, Alpine Function, Levy Function, Weierstrass Function, Michalewicz Function, Dixon & Price Function). The solution values obtained for each of the selected functions were compared with the solutions obtained using the Harris Hawks Optimizer (HHO), the Charged System Search Algorithm (CSS), and the Backtracking Search Optimization Algorithm (BSA).

Keywords: "Crayfish optimization algorithm (COA), Harris hawks optimizer (HHO), charged system search algorithm (CSS), backtracking search optimization (BSA), fixed and variable size unimodal and multimodal test functions."

1. Giriş

Son yıllarda, sorunları çözmenin giderek artan karmaşıklığı ve nihai sonuçların belirsizliği ile daha iyi optimizasyon algoritmalarına olan talep artmaktadır. Optimizasyon problemleri mühendislikte ve karar verme süreçlerinde çok yaygın bir şekilde kullanılmaktadır. Bir problemin optimizasyonu, verilen kısıtlar altında o problem için mevcut olan çözümler arasında en iyi olana ulaşmak demektir, kısaca en iyi çözümü bulmaktır. Bu tür problemlerin çözümü için önceleri deterministik (belirli) yöntemleri kullanan geleneksel teknikler kullanılırken, son yıllarda sıklıkla deterministik olmayan yöntemleri kullanan meta-sezgisel algoritmalar geliştirilmiştir [1-5].

Meta-sezgisel optimizasyon algoritmaları optimal çözümü elde etmek için genellikle iteratif çözüm sürecinde çözüm aralığında birden fazla çözümle başlarlar. Bu tür algoritmalarda her iterasyonda çözümler, yalnızca amaç fonksiyonuna bağlı olarak sezgisel tabanlı karar alma ile güncellenirler. Elde edilen tüm çözümler belirli bir bölgeye çok yakınsa, algoritmanın yerel optimizasyona takılması ve daha iyi bir çözüm elde edememesi durumu ortaya çıkar. Güncelleme formülü değeri çok büyük veya çok küçük olduğunda çözümün arama kapsamından çıkması söz konusu olabilir. Bu durumda algoritmanın optimizasyon performansı düşecektir. Yerel optimizasyon, belirli bir aralıktaki çözümün minimumunu veya maksimumunu ifade eder. Bu ifade tüm bölgenin minimumu veya maksimumu anlamına gelmez. Küresel optimum ise tüm bölgede çözümün minimumunu veya maksimumunu ifade eder [6].

Geleneksel ile meta-sezgisel algoritmalar karşılaştırıldığında, meta-sezgisel algoritmaların anlaşılması ve uygulanması daha kolay olduğu görülür. Geleneksel optimizasyon algoritmaları, hesaplama karmaşıklığı ve yakınsama açısından teorik olarak

[1] Corresponding Author
E-mail Address: celal_yasar@ksbu.edu.tr

analiz edilebilir. Meta-sezgisel algoritma, yerel optimumdan atlama ve etkili tasarım yoluyla bir noktaya yakınsama arasında iyi bir dengeye sahip olabilir, bu da belirli bir optimizasyon problemi için küresel optimumu bulma olasılığını artırır. Meta-sezgisel algoritmalar çözüm alanından ve başlangıç koşullarından bağımsız olarak ve ilave olarak güçlü sağlamlığa sahip oldukları için optimizasyon probleminin küresel en iyi çözümünü bulabildiklerinden pratikte yaygın olarak kullanılmaktadır. Bu nedenle günümüzde literatürde doğanın özelliklerinden esinlenerek geliştirilmiş birçok türde algoritmaya rastlanmaktadır. Bu algoritmalar sürü zekâsı, fizik tabanlı, evrimsel ve insandan ilham alan algoritmalar olarak dört kategoriye ayrılabilir [7].

Doğada bulunan hayvanların davranışlarından esinlenen algoritmalara sürü zekâsı algoritmaları denir. Bu grupta literatürde bulunan algoritmalardan bazıları, parçacık sürü optimizasyonu (particle swarm optimization, PSO) [8] ve yapay arı kolonisi algoritması (artificial bee colony algorithm, ABC) [9] örnek olarak verilebilir.

Fiziksel olaylardan esinlenen algoritmalara fizik tabanlı algoritmalar denir. Fizik tabanlı algoritmalar alanında birçok temsili çalışma bulunmaktadır. Bu gruba ait literatürde bulunan algoritmalardan bazıları ise, yerçekimi arama algoritması (gravitational search algorithm, GSA) [10] ve termal değişim optimizasyonu (thermal exchange optimization, TEO) [11] örnek olarak sayılabilir.

Evrimsel algoritmalar, doğanın evrim mekanizmasını taklit eden bir tür meta-sezgisel algoritmadır. Bu gruba ait literatürde bulunan algoritmalardan bazıları, genetik algoritmalar (genetic algorithm, GA) [12], geri izleme arama optimizasyon (backtracking search optimization algorithm BSA) [13] ve farksal gelişim (differential evolution, DE) [14] algoritması örnek olarak verilebilir.

İnsandan esinlenen algoritmalar ise, insan davranış alışkanlıklarından esinlenen meta-sezgisel algoritmalardır. Bu gruba ait literatürde bulunan algoritmalardan bazıları ise, harmoni arama (harmony search, HS) [15] ve öğretim öğrenme tabanlı optimizasyon (teaching learning based optimization, TLBO) [16] örnek olarak sayılabilir.

Son yıllarda literatüre birçok meta-sezgisel algoritma kazandırılmış olsa da bütün algoritmalar tüm problemleri çözer denilemez. Bir algoritma bir problemde çok iyi bir etkiye sahip olduğunda, diğer problemlerde iyi bir etkiye sahip olmayabilir. Bu nedenle her geçen gün yeni meta sezgisel algoritmalar üretilerek literatüre kazandırılmaya devam edilmektedir. Bu çalışmada son dönemde Jia ve arkadaşları tarafından geliştirilen kerevitin beslenme davranışı ve yazlık davranışına dayalı yeni bir meta-sezgisel optimizasyon algoritması olan kerevit optimizasyon algoritmasının (crayfish optimization algorithm - COA) [7] çeşitli test fonksiyonları üzerlerindeki çözüm bulma yeteneği ve performansı araştırılmıştır.

Bu çalışmada COA algoritması toplamda on adet sabit ve değişken boyutlara sahip unimodal ve multimodal test fonksiyonuna uygulanmıştır. COA'nın uygulanmasıyla elde edilen sonuçları karşılaştırmak için literatürden üç farklı algoritma seçilerek aynı fonksiyonlara aynı şartlarda uygulanmıştır. Çalışmada sürü tabanlı olan haris şahini optimize edici (harris hawks optimizer-HHO) [17, 18], fizik tabanlı olan yüklü sistem arama algoritması (charged system search algorithm-CSS) [19, 20] ve evrim tabanlı olan geri izleme arama optimizasyon (backtracking search optimization algorithm-BSA) [13] algoritması seçilmiştir. Sonuçları karşılaştırma için ele alınan bu algoritmalardan, HHO algoritması, küresel optimum arayışında etkili keşif ve sömürü aşamaları sunması nedeniyle literatürde birçok zorlu optimizasyon probleminde başarılı bulunmuştur. Özellikle hızlı yakınsama ve yüksek çözüm hassasiyeti sağlaması güçlü yönlerindendir. Ancak, HHO bazı durumlarda erken yakınsama (premature convergence) problemine yatkın olabilir ve çok karmaşık arama alanlarında çeşitliliği korumakta zorlanabilir. CSS algoritması ise, güçlü fiziksel temelli modellemesi sayesinde yüksek keşif kapasitesi ve iyi küresel arama yeteneği sunar. Özellikle başlangıç aşamalarında çözüm alanında geniş kapsamlı araştırma yapabilmesi kuvvetli bir yönüdür. Ancak, literatürde CSS'in işlem süresi açısından diğer meta-sezgisel algoritmalara göre daha maliyetli olabileceği ve parametre ayarlarının sonuçlar üzerinde önemli etkiler yaratabileceği belirtilmiştir. BSA'nın ise güçlü ve zayıf yönleri incelenirse, hafif yapısı ve düşük parametre bağımlılığı nedeniyle hızlı uygulama ve düşük hesaplama maliyeti avantajlarına sahiptir. Ayrıca, yerel optimumlardan kaçınma kabiliyeti yüksektir. Bununla birlikte, özellikle çok karmaşık, yüksek boyutlu problemlerde, keşif ve sömürü dengesi açısından diğer gelişmiş algoritmalara kıyasla daha düşük performans sergileyebileceği değerlendirilmektedir.

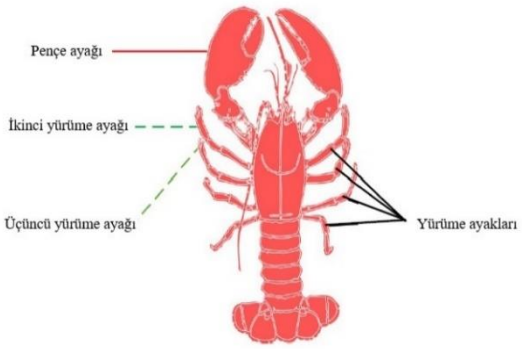
2. Kerevit Optimizasyon Algoritması (Crayfish Optimization Algorithm - COA)

2.1. Doğadaki Kerevit ve ilham Kaynağı

Kerevit karidese benzer ve kabuğu çok serttir. Hayvan taksonomisinde Arthropoda, Crustacea ve Decapoda sınıflarına ait olup genellikle tatlı su habitatının önemli bir türü olarak kabul edilir. Kerevitler göller, nehirler, akarsular, geçici göletler ve sadece mevsimsel olarak ıslak kalan alanlara kadar çok çeşitli tatlı su habitatlarında yaşarlar. Kerevitlerin çukur kazma ve geceleri ortaya çıkma alışkanlığı vardır. Dünyada kaydedilen 600'den fazla tatlı su kereviti türü olup bunlar coğrafi konum veya habitattan bağımsız olarak çukur kazma yeteneğine sahiptir. Bütün kerevitlerin mağaraları vardır ve bu mağaraların şekli ve işlevi türlere göre değişir. Bu değişiklik, kerevitlerin mağara ekolojisine dayalı olarak sınıflandırılmasından elde edilmiştir. Kerevit, doğal düşmanlardan ve yaz sıcağından kaçınmak için çukur kazar. Ayrıca, yırtıcılardan kaçınmak, vücudunun kurumasını önlemek, yiyecek aramak ve kuluçka bakımını sağlamak için de mağaralar kullanılabilir. Kerevit, belirli bir sıcaklık

aralığında yaşayan su hayvanıdır. Çeşitli hava koşullarına uyum sağlayabilir, ticari olarak pelet yemi sağlayabilir ve ayrıca ucuza yetiştirilebilir. Altı aylık beslenmeden sonra kerevit 40-60 gr'a kadar büyüyebilir. Kerevit, tüm yaprak döken canlılar gibi, yaşam ortamındaki sıcaklık değişiklikleri kerevitlerin büyüme oranlarını uzatır veya kısaltır. Bu nedenle, sıcaklığın kerevitlerin hayatta kalması üzerinde önemli bir etkisi olmuştur. Kerevitler 11 °C ile 44 °C arasında bu canlı hayatta kalabilir, ancak bu durum kerevitlerin büyümesi için uygun olduğu anlamına gelmez. Kerevitlerin büyümesi için uygun sıcaklık 15 °C ila 30 °C'dir. Kerevitlerin hayatta kalması için en uygun sıcaklık 25 °C'dir. Su sıcaklığı çok düşükse, bu durum kerevitlerin daha az besin yemelerine veya hiç besin yememelerine neden olur. Böylece bu durum kerevitlerin büyümelerini etkiler ve hatta ölümlerine neden olabilir. Su sıcaklığının çok yüksek olması da, kerevitlerin oksijensiz olarak karaya çıkmalarına ve hatta ölümlerine neden olur. Kerevitin şematik diyagramı Şekil 1'de gösterilmiştir. Kerevit, her türlü tahıl, sebze, karasal otlar, su kütlelerindeki su bitkileri, epifitik algler, zooplanktonlar, suda yaşayan böcekler, küçük benthos ve hayvan leşleri ile beslenebilen omnivor bir hayvandır. Kerevitler avlanırken genellikle pençeleriyle büyük yiyecekleri yakalarlar, tutmak ve kemirmek için 2. ve 3. yürüme ayaklarına göndermeden önce parçalarlar. Küçük yiyecekler için ise doğrudan tutmak ve ısırmak için ikinci ve üçüncü yürüme ayaklarını Şekil 2'de gösterildiği gibi kullanırlar. Avlandıktan sonra, genellikle hızlı bir şekilde saklanırlar veya diğer kerevitler tarafından soyulmaktan korunmak için kıskaçlarını kullanırlar [7].

Şekil 1. Kerevitin şematik diyagramı [7].



Şekil 2. Kerevitin yapısal diyagramı [7].

Kerevitlerin belirtilen bu özellikleri temel alınarak, yeni bir sürü zekâsı tabanlı optimizasyon algoritması olan COA oluşturulmuştur. Bu algoritma kerevitlerin yazlık tatil yeri arama, birbirleriyle rekabet etme ve besin arama gibi davranışlarına dayalı matematiksel bir model kurar. Diğer optimizasyon algoritmaları ile karşılaştırıldığında, COA, kerevitlerin yazlık tatil köyü mekanizmasını, rekabet mekanizmasını ve besin arama mekanizmasını simüle etmesi bakımından özeldir. COA ayrıca kerevitlerin büyük yiyecekleri parçalama sürecini de simüle etmiştir [7]. Aşağıdaki bölümlerde verilmiş olan 10 adet farklı matematiksel test fonksiyonlarının çözümüne ilişkin sonuçlardan, COA'nın mevcut meta-sezgisel algoritmalara kıyasla büyük avantajlara sahip olduğu görülmüştür.

2.2. İlham Kaynağı

COA, kerevitlerin yiyecek arama, yaz tatili için yer arama ve rekabetçi davranışlarından ilham alarak oluşturulmuştur. Yiyecek arama aşaması ve rekabet aşaması COA'nın kullanım (explotation) aşamasıdır ve yaz tatili aşaması ise COA'nın keşif aşamasıdır. Sürü zekâsı optimizasyonunun özelliklerini yansıtmak için, kerevit kolonisi (X), algoritmanın ilk aşamasında tanımlanır. X_i, bir adet çözümü gösteren *i.* kerevitin konumudur. X_i, uygunluk değeri olarak da bilinen bir çözüm elde etmek için $f(x)$ fonksiyonunu oluşturur. COA'nın keşfi ve kullanılması, bireyin bulunduğu ortamın sıcaklığını temsil eden rastgele bir sabit olan sıcaklık tarafından düzenlenir. Sıcaklık çok yüksek olduğunda, COA yaz tatili aşamasına veya rekabet aşamasına girecektir. Yaz tatili aşamasında, yeni çözüm bireysel konum olan X_i ve mağara konumu olan X_{shade}'e göre güncellenir. Sıcaklık uygun olduğunda, COA yiyecek arama aşamasına girecektir. Yiyecek arama aşamasında, yiyeceğin yeri en iyi yer ve aynı zamanda en uygun çözüm olarak da bilinir. Yiyeceğin boyutu, mevcut çözüm uygunluğu ve optimal çözüm $fitness_{food}$ ile elde edilir. Besin uygun olduğunda, konumları X_i, yiyecek alım sabiti p ve yiyecek pozisyonu X_{food} güncellemelerine göre kerevitler yeni çözümler üretirler. Besin çok büyük olduğunda, kerevit yiyeceğini parçalamak için pençe ayağını kullanacak ve ardından dönüşümlü olarak yemek için ikinci ve üçüncü yürüme ayaklarını kullanacaktır. Kerevitlerde besin alımı kontrol edilir. Yiyecek alımı sıcaklık ile belirlenir ve pozitif bir dağılım gösterir [7].

2.3. Popülasyonu Başlatma

Çok boyutlu optimizasyon probleminde, her kerevit (1**dim*'lik) bir matristir. Her sütun matrisi bir problemin çözümünü temsil eder. Bir dizi değişkende ($X_{i,1}$, $X_{i,2}$, …, $X_{i,dim}$), her X_i değişkeni üst ve alt sınırlar arasında yer almalıdır. COA'nın başlatılması, uzayda rastgele bir grup aday çözüm için X oluşturmaktır. Aday çözüm X, popülasyon büyüklüğü N ve boyut sayısı olan *dim*'e dayanmaktadır. COA algoritmasının başlatılması Denklem (1)'de gösterilmiştir [7].

$$X = \left[X_1, X_2, ..., X_N\right] = \begin{bmatrix} X_{1,1} \cdots X_{1,j} \cdots X_{1,dim} \\ \vdots \quad \cdots \quad \vdots \quad \cdots \quad \vdots \\ X_{i,1} \cdots X_{i,j} \cdots X_{i,dim} \\ \vdots \quad \cdots \quad \vdots \quad \cdots \quad \vdots \\ X_{N,1} \cdots X_{N,j} \cdots X_{N,dim} \end{bmatrix} \tag{1}$$

Burada X başlangıç popülasyon konumunu, N popülasyon sayısını ve *dim* ise popülasyon boyutunu göstermektedir. Denklemde $X_{i,j}$ ise *i.* Bireyin *j.* boyutundaki konumunu göstermekte olup değeri Denklem (2) kullanılarak bulunur.

$$X_{i,j} = lb_j + (ub_j - lb_j) \times rand \tag{2}$$

Denklemde lb_j ifadesi *j.* boyutun alt sınırını temsil ederken, ub_j ise *j.* boyutun üst sınırını temsil eder ve *rand* ise rastgele bir sayıdır.

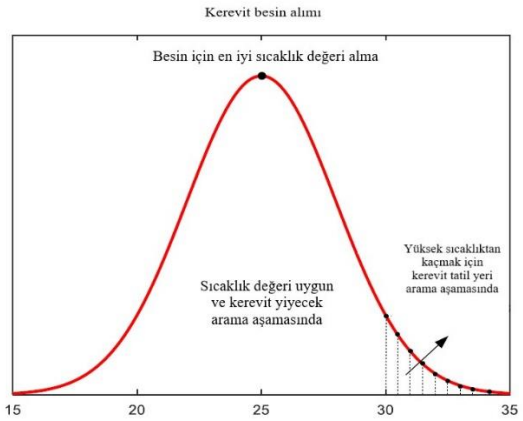
2.4. Kerevitlerin Sıcaklığını ve Besin Alımını Tanımlama

Sıcaklığın değişmesi kerevitin davranışını etkileyecek ve onun farklı aşamalara girmesini sağlayacaktır. Sıcaklık, Denklem (3)'te tanımlanmıştır. Sıcaklık 30 ˚C'den yüksek olduğunda, kerevit yaz tatili için serin bir yer seçecektir. Kerevit sıcaklık değerleri uygun olduğunda yiyecek arama davranışı gerçekleştirecektir. Kerevitin beslenme miktarı sıcaklıktan etkilenir. Kerevitin sıcaklık açısından beslenme aralık değerleri 15 ˚C ile 30 ˚C olup en iyi değer 25 ˚C'dir. Bu nedenle, kerevitin beslenme miktarı normal dağılıma yaklaştırılabilir, böylece beslenme miktarı sıcaklıktan etkilenir. Kerevitlerin 20 ˚C ila 30 ˚C arasında güçlü yiyecek arama davranışı olduğundan, COA 20 ˚C ila 35 ˚C arasında bir sıcaklık aralığı tanımlar. Kerevitlerin besin alımının matematiksel modeli Denklem (4)'te ve besin alımının şematik diyagramı ise Şekil 3'te gösterilmiştir [7].

$$temp = rand \times 15 + 20 \tag{3}$$

$$p = C_1 \times \left(\frac{1}{\sqrt{2 \times \pi} \times \sigma} \times exp\left(- \frac{(temp - \mu)^2}{2\sigma^2} \right) \right) \tag{4}$$

Denklemlerde, *temp* kerevitin bulunduğu ortamın sıcaklığını, μ kerevit için en uygun sıcaklık değerini temsil etmektedir. Denklemlerde, σ ve C_1 ise kerevitlerin farklı sıcaklıklarda besin alımını kontrol etmek için kullanılır.



Şekil 3. Kerevitin besin alımı üzerinde ortam sıcaklığının etkisi [7].

2.5. Yaz Tatili Beldesi Aşaması (Keşif)

Keşif aşamasında *temp>30* olduğunda, sıcaklık çok yüksek olup kerevit yaz tatili için mağaraya girmeyi seçecektir. Mağara X_{shade} Denklem (5)'te tanımlanmıştır. Denklemde, X_G iterasyon sayısıyla şimdiye kadar elde edilen en uygun konumu temsil eder ve X_L mevcut popülasyonun en uygun konumunu temsil eder [7].

$$X_{shade} = \left(X_G + X_L\right) / 2 \tag{5}$$

Kerevitlerin mağaralar için mücadele etmesi rastgele bir olaydır. Şekil 4.A'da gösterildiği gibi *rand<0,5* olursa, bu mağaralar için yarışan başka bir kerevit olmadığı ve kerevitlerin yaz tatili için doğrudan mağaraya gireceği anlamına gelir. Bu anda, kerevit Denklem (6) kullanılarak yazlık tatil beldesi için mağaraya girecektir [7].

$$X_{i,j}^{t+1} = X_{i,j}^{t} + C_2 \times rand \times \left(X_{shade} - X_{i,j}^{t} \right) \tag{6}$$

Burada, *t* geçerli iterasyon sayısını ve *t+1* ise yeni nesil iterasyon sayısını temsil eder, C_2 ise Denklem (7)'de gösterildiği gibi azalan bir eğridir. Burada *T*, maksimum iterasyon sayısını göstermektedir.

$$C_2 = 2 - \left(t / T \right) \tag{7}$$

Yazlık tatil beldesi aşamasında (keşif) kerevitin amacı, en uygun çözümü temsil eden mağaraya yaklaşmaktır. Bu noktada kerevit mağaraya yaklaşacaktır. Bu, bireyleri en uygun çözüme yaklaştırır ve COA'dan yararlanma yeteneğini artırır ve böylece algoritmanın daha hızlı bir şekilde yakınsamasını sağlar [7].

2.6. Yarışma Aşaması (Kullanım)

Şekil 4.B'de gösterildiği gibi bu aşamada *temp>30* ve *rand≥0,5* olduğunda diğer kerevitlerin de mağaraya ilgi duyduğu ve mağarayı ele geçirmek için birbirleriyle mücadele edecekleri anlamına gelir. Kerevitin mağara için rekabeti Denklem (8)'le ifade edilmiştir [7].

$$X_{i,j}^{t+1} = X_{i,j}^{t} - X_{z,j}^{t} + X_{shade} \tag{8}$$

Burada *z* Denklem (9)'da gösterildiği gibi kerevitin rastgele bir bireyini temsil eder.

$$z = round\left(rand \times \left(N - 1 \right) \right) + 1 \tag{9}$$

Yarışma aşamasında, kerevitler birbirleriyle yarışırlar. Kerevit kendi X_i konumunu, diğer bir kerevitin X_s konumuna göre ayarlar. Bu şekilde konumlar ayarlanarak, COA'nın arama aralığı genişletilir ve böylelikle algoritmanın keşif yeteneği geliştirilmiş olur.

(A) **(B)**

Şekil 4. A) Kerevitler doğrudan mağaraya girer. B) Kerevitler mağaraya sahip olmak için birbirleriyle yarışırlar [7].

2.7. Besin Arama Aşaması (Sömürü)

Bu aşamada *temp≤30* olduğunda, sıcaklık kerevitlerin beslenmesi için uygundur ve bu aşamada kerevit yiyeceğe doğru hareket edecektir. Yiyeceği bulduktan sonra, kerevit yiyeceğin büyüklüğünü değerlendirecektir. Eğer yiyecek çok büyükse, kerevit yiyeceği pençeleriyle yırtacak ve dönüşümlü olarak ikinci ve üçüncü yürüme ayaklarıyla yiyecektir. Yiyecek konumu X_{food} olarak Denklem (10)'a göre tanımlanır [7].

$$X_{food} = X_G \tag{10}$$

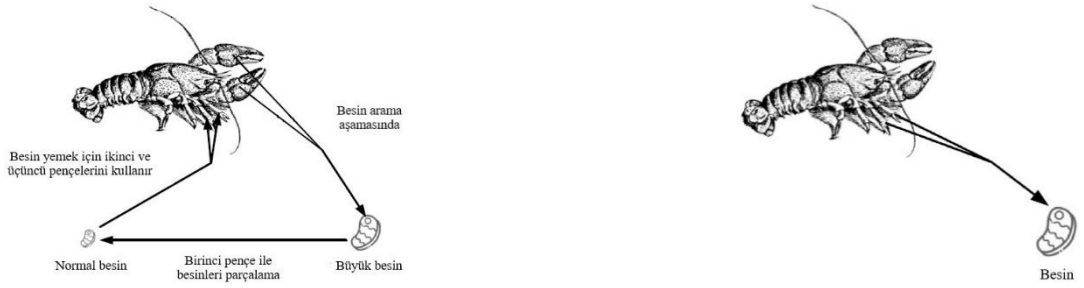
Gıda boyutu ise *Q* olarak Denklem (11)'e göre tanımlanmaktadır.

$$Q = C_3 \times rand \times \left(fitness_i \,/\, fitness_{food} \right) \tag{11}$$

Denklemde C_3 besin faktörüdür ve en büyük besini temsil eder ve değeri sabit olarak 3'tür, $fitness_i$, birinci kerevitin uygunluk değerini ve $fitness_{food}$ ise besin konumunun uygunluk değerini temsil eder.

Kerevitin yiyecek boyutuna ilişkin durumu en büyük yiyeceğin boyutundan gelir. $Q>(C_3+1)/2$ olduğunda, bu yiyeceğin çok büyük olduğu anlamına gelir. Bu durumda Şekil 5.A'da gösterildiği gibi kerevit ilk pençe ayağıyla yiyeceği yırtacaktır. Bu yiyeceğe ait matematiksel ifade Denklem (12)'de tanımlanmıştır [7].

$$X_{food} = \exp\left(-\frac{1}{Q}\right) \times X_{food} \tag{12}$$



(A) **(B)**

Şekil 5. A) Kerevit yemekten önce yiyecekleri parçalar. B) Kerevit doğrudan yemeğini yer [7].

Yiyecek parçalandığında ve küçüldüğünde, ikinci ve üçüncü pençeler dönüşümlü olarak yiyeceği alır ve ağza koyar. Alternatif işlemi simüle etmek için ise sinüs fonksiyonu ve kosinüs fonksiyonunun kombinasyonu kullanılır. Ayrıca Şekil 5.B'de gösterildiği kerevit tarafından elde edilen yiyecek aynı zamanda gıda alımı ile de ilgilidir, bu nedenle yiyecek arama eşitliği Denklem (13)'te verilmiştir [7].

$$X_{i,j}^{t+1} = X_{i,j}^{t} + X_{food} \times p \times \left(\cos\left(2 \times \pi \times rand\right) - \sin\left(2 \times \pi \times rand\right)\right) \tag{13}$$

$Q \leq (C_3+1)/2$ olduğunda, kerevitin sadece yiyeceğe doğru hareket etmesi ve doğrudan yemesi gerekir, bu durum ise Denklem (14)'te gösterilmiştir.

$$X_{i,j}^{t+1} = \left(X_{i,j}^{t} - X_{food}\right) \times p + p \times rand \times X_{i,j}^{t} \tag{14}$$

Besin arama aşamasında, kerevitler yiyeceklerinin Q boyutuna bağlı olarak farklı besleme yöntemleri kullanırlar ve yiyecek X_{food} en uygun çözümü temsil eder. Besinin Q boyutu kerevitin gıda yemesi için uygun olduğunda, kerevit yiyeceğe yaklaşacaktır. Q değerinin çok büyük olması, kerevit ile optimal çözüm arasında önemli bir fark olduğunu gösterir. Bu nedenle X_{food} azaltılmalı ve besine yaklaştırılmalıdır. Ayrıca kerevit için besin alımı geliştirme algoritmasının rastgeleliği kontrol edilmelidir. Besin arama aşaması boyunca, COA en uygun çözüme yaklaşacak, algoritmanın yararlanma yeteneğini geliştirecek ve iyi bir yakınsama yeteneğine sahip olmasını sağlayacaktır [7].

2.8. Algoritma Uygulaması

Adım 1: Parametre tanıtımı ve popülasyonun başlatılması

İterasyon sayısı *T,* popülasyon sayısı *N,* boyut *dim,* üst sınır *ub* ve alt sınır *lb'yi* tanımlayarak üst ve alt sınırlara dayalı olarak popülasyon *X*'i başlatın. Popülasyon başlatma adımı Denklem (2)'den elde edilir [7].

Adım 2: Sıcaklık tanımı

Kerevitin bulunduğu ortam sıcaklığı, COA'nın farklı aşamalara girmesini sağlamak için Denklem (3)'e göre tanımlanır.

Adım 3: Yaz tatili aşaması ve yarışma aşaması

3.1. *temp>30* ve *rand<0,5* olduğunda, COA yaz tatili aşamasına girer. Bu sırada COA, mağara konumuna (X_{shade}) ve kerevit konumuna ($X_{i,j}^{t}$) göre yeni bir ($X_{i,j}^{t+1}$) konumu elde eder. Güncellenmiş formül Denklem (6)'daki gibi olur ve sonra 5. adıma geçilir.

3.2. *temp>30* ve *rand≥0,5* olduğunda, COA rekabet aşamasına girer. Bu aşamada, iki kerevit Denklem (8)'e göre mağara için rekabet edecek ve mağara pozisyonuna (X_{shade}) ve iki kerevitin pozisyonuna ($X_{i,j}^{t}$, $X_{z,j}^{t}$) göre yeni bir ($X_{i,j}^{t+1}$) pozisyonu elde edecektir ve sonrasında 5. adıma geçilir.

Adım 4: Yiyecek arama aşaması

4.1. *temp≤30* olduğunda, COA yiyecek arama aşamasına girer, yiyecek alımı *p* ve yiyecek boyutu *Q* olmak üzere Denklem (4) ve (11)'e göre tanımlanır.

4.2. $Q>(C_3+1)/2$ olur ise, besin Denklem (12)'ye göre parçalanır. Bundan sonra ise Denklem (13) aracılığıyla yeni konum elde edilir ve adım 5'e geçilir.

4.3. $Q\leq(C_3+1)/2$ olur ise, Denklem (14) aracılığıyla yeni bir konum elde edilir ve 5. adıma geçilir.

Adım 5: Değerlendirme aşaması

Bu aşamada popülasyon değerlendirilir ve döngüden çıkıp çıkılmayacağına karar verilir. Değilse, 2. adıma geri dönülür.

Adım 6: En iyi $fitness_{best}$ değerini çıkarma

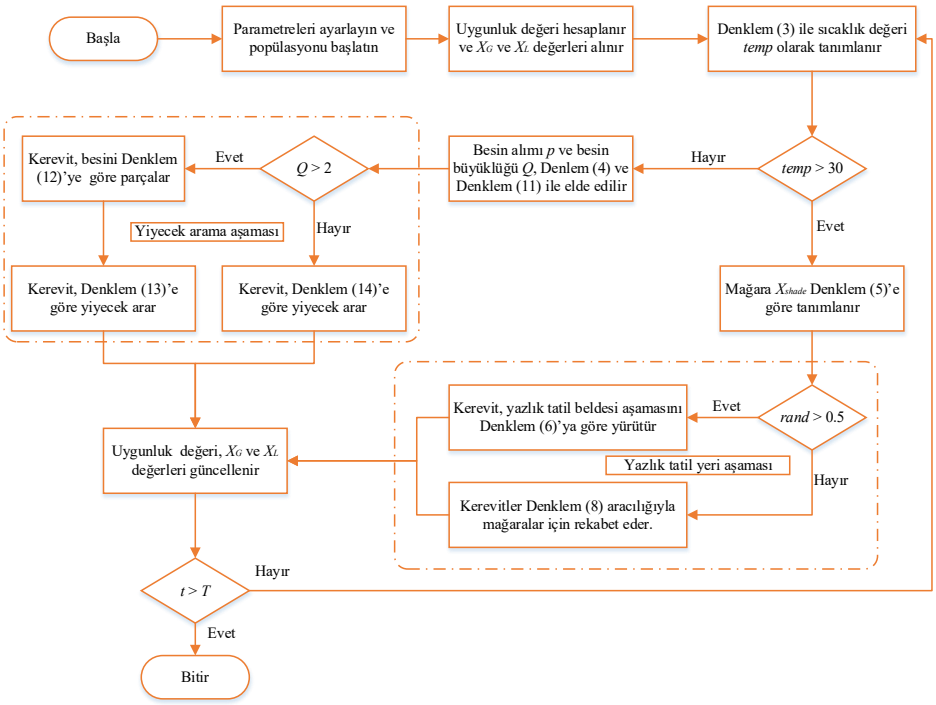
2.9. COA'nın Sözde Kodu ve Akış Şeması

COA'nın sözde kodu aşağıdaki gibidir [7]:

Algoritma 1. COA Algoritmasının sözde kodu [7]

```
Başlatma için iterasyon sayısı T, popülasyon sayısı N ve boyut sayısı dim belirleme.
Rastgele bir başlangıç popülasyonu oluşturma.
X_G ve X_L değerleri elde etmek için popülasyonun uygunluk değerini hesaplama.
While t<T
   Sıcaklığın Denklem (3) ile tanımlanması.
   If temp>30
      Mağara yeri için X_shade'i Denklem (5)'e göre tanımlama.
      If rand<0,5
         Kerevit, yazlık tatil yeri arama aşamasını Denklem (6)'ya göre yürütür.
      Else
         Kerevitler, Denklem (8) aracılığıyla mağaralar için rekabet eder.
      End
   Else
      Besin alımı p ve besin büyüklüğü Q, Denklem (4) ve Denklem (11) ile elde edilir.
      If Q>2
         Kerevit yiyecekleri Denklem (12) ile parçalar.
         Kerevit, Denklem (13)'e göre yiyecek arar.
      Else
         Kerevit, Denklem (14)'e göre yiyecek arar.
      End
   End
   X_G ve X_L değerleri ile uygunluk değerlerini güncelleme.
   t=t+1
End
```

COA'nın akış şeması Şekil 6'da gösterilmiştir [7].



Şekil 6. COA'nın akış şeması [7].

3. Sonuçları Karşılaştırmak için Seçilen Diğer Algoritmalar

Bu bölümde COA algoritması uygulanarak elde edilen sonuçları karşılaştırmak için kullanılan HHO, CSS ve BSA algoritmalarının yapıları orijinal kaynaklarından alıntılanarak kısaca özetlenmiştir.

3.1. Harris Şahin Optimizasyonu (HHO) Algoritması

Bu algoritma, doğada iyi bilinen bir yırtıcı kuş olan Harris şahinlerinin avlanma stratejisi benzetimi yapılarak oluşturulmuştur [17, 18]. Harris şahinleri, diğer yırtıcı kuşlardan farklı olarak genellikle aile üyeleriyle birlikte bir avı keşfetmek ve yakalamak için saldırırlar. Bu kuş, potansiyel avı izleme, çevreleme, temizleme ve sonunda saldırma konusunda evrimleşmiş yenilikçi takım kovalama yetenekleri gösterir. Bu akıllı kuşlar, üreme dışı mevsimde birkaç bireyden oluşan akşam yemeği partileri düzenleyebilirler. Yırtıcı kuşlar aleminde gerçek anlamda işbirlikçi avcılar olarak bilinirler. Harris şahinleri avlanma süreçlerinde özellikle tavşan avlama sürecinde sürü olarak hareket ederler. Sürünün bir lideri (en iyi şahin) bulunur, lider ve sürünün diğer üyeleri öncelikle keşif uçuşları yaparlar. Harris şahinleri, koşulların dinamik doğasına ve bir avın kaçış modellerine bağlı olarak çeşitli kovalama stratejileri gösterebilirler. Bir değiştirme taktiği, lider şahin avın üzerine eğilip kaybolduğunda ve kovalamaca parti üyelerinden biri tarafından sürdürüldüğünde gerçekleşir. Bu durum kaçan tavşanı şaşırtır, tavşanı yorar ve onun savunmasızlığını artırır. Son olarak, genellikle en güçlü ve deneyimli olan şahinlerden biri yorgun tavşanı zahmetsizce yakalayıp diğer parti üyeleriyle paylaşır. Harris şahinlerinin bu özellikleri Heidari ve arkadaşları tarafından 2019 yılında haris şahinleri optimizasyonu (harris hawks optimization, HHO) algoritması olarak modellenmiştir [17]. HHO popülasyon temelli bir optimizasyon tekniği olup keşif, keşiften saldırıya geçiş ve saldırı aşamalarından oluşur. HHO'da keşif aşamasında iki farklı strateji olarak uygulanır. Keşiften saldırıya geçiş aşamasında, avın kaçma sırasında enerjisinin azalması durumuna karşılık Harris şahinleri farklı saldırı davranışları arasında geçişler yapabilirler. Bu nedenle saldırı aşaması algoritmada yumuşak, sert, kademeli hızlı dalışlarla yumuşak ve kademeli hızlı dalışlarla sert kuşatma olmak üzere dört farklı strateji olarak modellenmiştir. HHO algoritmasının sözde kodu Algoritma 2'de verilmiştir [18].

Algoritma 2. HHO Algoritmasının sözde kodu [17, 18]

Inputs: N popülasyon büyüklüğü ve maksimum iterasyon sayısı T.
Outputs: Tavşanın konumu ve tavşanın *fitness* değeri.
Rastgele bir popülasyonu başlat. $X_i(i=1,2,...,N)$.
while (durdurma kriteri sağlanmadıysa) **do**
  Şahinlerin *fitness* değerlerini hesapla.
  X_{rabbit} tavşanının konumunu ayarla (en iyi konum olarak).
  for (her şahin (X_i)) **do**
    Başlangıç enerjisi E_0 , sıçrama gücü J ve E değerini güncelle.
    if ($|E|\geq1$) **then** Keşif Aşamasındaki Konum vektörünü güncelle.
    if ($|E|<1$) **then** (Sömürü Aşaması)
      if (r≥0,5 ve $|E|\geq0,5$) **then** Yumuşak Kuşatma Konum vektörünü güncelle.
      else if (r≥0,5 ve $|E|<0,5$) **then** Sıkı Kuşatma Konum vektörünü güncelle.
      else if (r<0,5 ve $|E|\geq0,5$) **then** İlerleyici Hızlı Dalmalarla Yumuşak Kuşatma Konum vektörünü güncelle.
      else if (r<0,5 ve $|E|<0,5$) **then** İlerleyici Hızlı Dalmalarla Sıkı Kuşatma Konum vektörünü güncelle.
Return X_{rabbit}

3.2. CSS Algoritması

Yüklü Sistem Arama (CSS) algoritması, Kaveh ve Talatahari tarafından geliştirilen bir meta-sezgisel optimizasyon yöntemidir. Algoritmanın temeli, Newton'un hareket yasaları ile elektrik fiziğindeki Coulomb ve Gauss yasalarına dayanır. Bu algoritmada, her bir ajan (çözüm adayı), elektrik yüklü bir parçacık (charged particle-CP) olarak ele alınır. Her bir CP, Coulomb ve Gauss yasalarına uygun şekilde diğer yüklü parçacıklar üzerine elektriksel kuvvet uygular ve bu yüklü bir küre şeklinde modellenir. Bu kuvvetin büyüklüğü, küre içindeki CP için CP'ler arasındaki mesafeyle orantılıyken, kürenin dışında bulunan bir CP için parçacıklar arasındaki mesafenin karesiyle ters orantılıdır ve bu kuvvetler çekici veya itici olarak ortaya çıkabilir. Arama alanındaki CP'lerin başlangıç konumları rastgele belirlenir. Genellikle çekici olarak çıkan kuvvet, CP'leri arama alanı içinde belirli bir alanda toplarken, itici kuvvet CP'leri dağıtmaya çalışır. Sonuç kuvvetleri ve hareket yasaları, CP'lerin yeni konumlarını belirler. Bu kurala göre, her CP, sonuç kuvvetleri ve önceki hızıyla yeni konumuna doğru hareket eder. Algoritma bu şekilde optimizasyon sürecinin ilerlemesini sağlar. Her CP, CP arama alanından çıkarsa, konumu, uyum aramasına dayalı yaklaşımla düzeltilir. Ayrıca, en iyi sonuçları kaydetmek için yüklü bellek kullanılır. CSS algoritmasının geliştirilmesi aşağıda kısaca özetlenen sekiz kural üzerinden gerçekleştirilmiştir. CSS algoritmasının bu kurallar çerçevesinde sözde kodu ise Algoritma 3'te verilmiştir [19, 20].

Kural 1: CSS bir dizi yüklü parçacığı (CP) dikkate alır ve her bir CP'nin bir yük büyüklüğü olup uzayı etrafında bir elektrik alanını oluşturur.

Kural 2: CP'lerin başlangıç konumları arama uzayında rastgele belirlenir.

Kural 3: Çekici kuvvetler cinsinden bakıldığında, herhangi bir CP başka biri CP'yi etkileyebilir, yani kötü bir CP iyi bir CP'yi etkileyebilir veya bunun tersi de geçerlidir. İyi bir CP kötü bir CP'yi de çekebilir.

Kural 4: Bir CP üzerinde etkili olan sonuç elektrik kuvvetinin değeri, ajanların birbirinden uzak olduğu ilk iterasyonda, bir CP üzerinde etkili olan sonuç kuvvetinin büyüklüğü, parçacıklar arasındaki ayrımın karesiyle ters orantılıdır. Fakat CP'lerin küçük bir alanda toplandığı ve CP'ler arasındaki ayrımın küçük olduğu durumlarda ise ayrım mesafesinin karesiyle ters orantılı olmak yerine parçacıkların ayrım mesafesiyle orantılı hale gelir.

Kural 5: Her bir CP'nin yeni konumu ve hızı her iterasyonda güncellenir.

Kural 6: Hesaplama maliyetini artırmadan algoritma performansını iyileştirebilmek için en iyi CP vektörlerini ve ilgili amaç fonksiyon değerlerini kayıt altına alabilen bir yüklü bellek (charged memory, CM) kullanılmalıdır.

Kural 7: Birçok meta-sezgisel algoritmada olduğu gibi CSS'de de iki büyük sorun vardır. Bu sorunlardan birincisi aramanın başında, sırasında ve sonunda keşif ve kullanım (exploitation) arasındaki denge ve ikincisi ise değişkenlerin sınırlarını ihlal eden bir etkenle nasıl başa çıkılacağı konusudur. İlk sorun, yukarıda belirtilen kuralların uygulanmasıyla doğal olarak çözülür. Ancak ikinci sorunu çözmek için en basit yaklaşımlardan biri, ihlal edilen değişken için en yakın sınır değerlerini kullanmaktır. İhlali gerçekleştiren parçacığı önceki konumuna geri döndürmek bazen zor olabilir. Bu durumda alternatif olarak değişken sınırlarını ihlal eden çözüm vektörünün herhangi bir bileşeni CM'den yeniden üretilebilir.

Kural 8: Algoritmayı sonlandırma kriterleri olarak maksimum yineleme sayısı, iyileştirme olmayan yineleme sayısı, minimum amaç fonksiyonu hatası ve son olarak en iyi ve en kötü CP'ler arasındaki fark işlemlerinden birisi kullanılır.

Algoritma 3. CSS Algoritmasının sözde kodu [19, 20]

Seviye 1: Başlatma
• **Adım 1:** *Başlatma.*
CSS algoritma parametrelerini başlatın; Rastgele konumlara ve ilişkili hızlara sahip Yüklü Parçacıklar dizisini başlatın (Kural 1 ve 2).
• **Adım 2:** *CP sıralaması.*
CP'ler için uygunluk fonksiyonunun değerlerini değerlendirin, birbirleriyle karşılaştırın ve artan şekilde sıralayın.
• **Adım 3:** *CM oluşturma.*
CM'deki ilk CP'lerin CMS numarasını ve hedef fonksiyonunun ilişkili değerlerini saklayın.
Seviye 2: Arama
• **Adım 1:** *Çekim kuvveti belirleme.*
Her bir CP'yi diğerlerine doğru hareket ettirme olasılığını belirleyin (Kural 3) ve her CP için çekim kuvveti vektörünü hesaplayın (Kural 4).
• **Adım 2:** *Çözüm oluşturma.*
Her bir CP'yi yeni konuma taşıyın ve hızları bulun (Kural 5).
• **Adım 3:** *CP konum düzeltmesi.*
Her CP izin verilen arama alanından çıkarsa, Kural 7'yi kullanarak konumunu düzeltin.
• **Adım 4**: *CP sıralaması.*
Yeni CP'ler için hedef fonksiyonunun değerlerini değerlendirin ve karşılaştırın ve bunları artan şekilde sıralayın.
• **Adım 5:** *CM güncellemesi.*
Bazı yeni CP vektörleri CM'deki en kötü vektörlerden daha iyiyse, daha iyi vektörleri CM'ye dahil edin ve en kötü olanları CM'den çıkarın (Kural 6).
Seviye 3: Sonlandırma kriteri kontrolü
• Sonlandırma kriteri karşılanana kadar arama seviyesi adımlarını tekrarlayın (Kural 8).

3.3. BSA Algoritması

BSA algoritması, optimizasyon problemlerinde yerel çözümlerden sıyrılarak küresel çözümler bulmayı hedefler. Algoritmanın işleyişi, başlangıç değerlerinin atanması, ilk seçim aşaması, mutasyon, çaprazlama ve son seçim aşaması gibi beş ana adıma dayanmaktadır. İlk değerin verilmesi aşamasında popülasyon büyüklüğü, problemin boyutu, popülasyon içerisindeki bir hedef birey ve son olarak da çözüm uzayındaki en alt ve en üst sınır değerleri belirlenir. Birinci seçim aşamasında arama yönünü hesaplamak için algoritmanın tarihsel popülasyonu belirlenir. Böylece, BSA algoritması geçmişte elde edilen değerleri, bir sonraki karar alma mekanizmasında kullanılmak üzere hafızaya alır. Tarihsel popülasyonunun belirlenmesiyle birlikte popülasyon üyeleri rastgele olarak yeniden sıralanır. Mutasyon aşamasında, mutant popülasyonunun ilk değerleri hesaplanır. Çaprazlama aşamasında, popülasyonun son durumu değerlendirmeye alınır. Bu aşamada, optimizasyon problemine göre yüksek performans gösteren popülasyon üyeleri, hedef popülasyon bireylerini belirlemek amacıyla seçilir. BSA algoritması ayrıca, mutasyon geçiren bireylerin çözüm uzayının dışına çıkmasını önlemek için bir sınırlandırma mekanizması kullanır. İkinci seçim aşamasında, popülasyon güncellenir ve en iyi birey seçilir. Her iterasyon döngüsünde, mevcut küresel en iyi değer tüm popülasyon bireyleriyle karşılaştırılır. Eğer herhangi bir bireyin amaç fonksiyonu değeri, mevcut küresel en iyiden daha üstünse, yeni küresel en iyi değer bu bireyin bulunduğu konum olur. BSA algoritmasının sözde kodu Algoritma 4'de verilmiştir [13].

Algoritma 4. BSA Algoritmasının sözde kodu [13]

Input: *ObjFun*, *N*, *D*, *maxcycle*, *mixrate*, $low_{1:D}$, $up_{1:D}$
Output: *globalminimum*, *globalminimizer*
// *rnd* ~ *U*(0,1), *rndn* ~ *N*(0,1), *w*=*rndint*(·), *rndint*(·) ~ U(1,·) | *w* ∈ {1, 2, 3,...,·*N*}
function bsa(*ObjFun*, *N*, *D*, *maxcycle*, *low*, *up*)
// BAŞLANGIÇ
globalminimum=inf
for *i* **from** 1 **to** *N* **do**
  for *j* **from** 1 **to** *D* **do**
    $P_{i,j}$=*rnd*(up_j-*lowj*)+*lowj* // Popülasyonun başlatılması, P.
    $oldP_{i,j}$=*rnd*(up_j-*lowj*)+*lowj* // oldP'nin başlatılması, oldP.
  end
  $fitnessP_i$=*ObjFun*(P_i) // P'nin ilk uygunluk değerleri.
end
for *iteration* **from** 1 **to** *maxcycle* **do**
  // SEÇİM-I
  if (*a*<*b* | *a*, *b* ~ *U*(0,1)) **then** *oldP*:=P **end**
  oldP:=*permuting*(*oldP*) // 'etkileme'; oldP içindeki iki bireyin pozisyonlarını rasgele değiştirme.

```
Deneme Popülasyonunun Oluşturulması
  // MUTASYON
  mutant=P+3·rndn·(oldP-P)
  // ÇAPRAZLAMA
  map1:N, 1:D=1 // Başlangıç haritası birlerden oluşan N x D matrisidir.
  if (c<d | c, d ~ U(0,1)) then
    for i from 1 to N do
      map i,u(1:[mixrate·rnd·D]= 0 | u=permuting(1,2,3,…,D)
    end
  else
    for i from 1 to N do, map i,randi(D)=0, end
  end
  // Deneme Popülasyonu Üretimi, T
  T:=mutant
  for i from 1 to N do
    for j from 1 to D do
      if map i,j=1 then T i,j:=P i,j
    end
  end
  // Sınır Kontrol Mekanizması
  for i from 1 to N do
    for j from 1 to D do
      if (T i,j<low j) or (T i,j>up j) then
        T i,j=rnd·(up j-low j)+low j
      end
    end
  end
  // Seçim-II
  fitnessT=ObjFun(T)
  for i from 1 to N do
    if fitnessT i<fitnessP i then
      fitnessP i:=fitnessT i
      P i=T i
    end
  end
  fitnessP best=min(fitnessP) | best ∈ {1,2,3,…,N}
  if fitnessP best<globalminimum then
    globalminimum:=fitnessPbest
    globalminimizer:=P best
    // Global minimum ve global minimizer dışarı aktarılır.
  end
end
```

4. Test Fonksiyonları

Bu çalışmada test fonksiyonu olarak değişken boyutlu fonksiyonlar kapsamında f_1 fonksiyonu unimodal (Elliptic Fonksiyon), f_2–f_7 arası fonksiyonlar multimodal (Non-Continuous Rastrigin, Alpine, Levy, Weierstrass, Michalewicz, Dixon&Price Fonksiyonları) ve sabit boyutlu fonksiyonlar kapsamında ise f_8-f_{10} arası fonksiyonlar (Schaffer, Himmelblau ve Kowalik Fonksiyonları) olmak üzere 10 adet test fonksiyonu seçilmiştir. Seçilen fonksiyonların grafikleri, temel özellikleri ve eşitlikleri aşağıdaki gibi Tablo 1'de gösterilmiştir.

Bu çalışmada yer alan test fonksiyonlarının COA ile çözümünde elde edilen sonuçlarını karşılaştırmak için HHO, CSS ve BSA algoritmaları kullanılmıştır. Algoritmaların performans, yakınsama, kararlılık ve hız açısından değerlendirilmesi için test fonksiyonlarının sonuçlarının değerlendirilmesi, test fonksiyonlarının boyutlarına göre 30, 50, 100 ve sabit boyutlu olmak üzere dört alt başlık altında ayrı ayrı yapılmıştır.

COA algoritmasının performansını en üst düzeye çıkarmak için etkili bir şekilde tahmin edilmesi gereken kerevitlerin farklı sıcaklıklarda besin alımını kontrol etmek için kullanılan σ ve C_1, kerevit için en uygun sıcaklık değerini ifade eden μ değeri ve besin faktörü olup en büyük besini temsil eden C_3 olmak üzere dört adet parametresi vardır. Yapılan çalışmalar sonucunda bu parametrelerin en iyi değerlerinin, σ için 3, C_1 için 0,2, μ için 25 ve C_3 için ise 3 olduğu sonucuna varılmıştır [7]. Çalışmada

iterasyon sayısı olarak tüm algoritmalar ve tüm test fonksiyonları için 500 (15000 fonksiyon çağırımı, *FCall*) alınmıştır. Bu çalışmada kullanılan tüm algoritmalara ait kullanılan parametre değerleri Tablo 2'de verilmiştir. Algoritmaların test fonksiyonlarına uygulanması için Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz işlemcili ve 6 GB RAM bellekli iş istasyonu kullanılarak MATLAB R2024a'da program geliştirilmiş ve her bir fonksiyon için 30'ar kez koşturulmuştur.

Tablo 1. *f1-f10* fonksiyonlarına ait özellikler

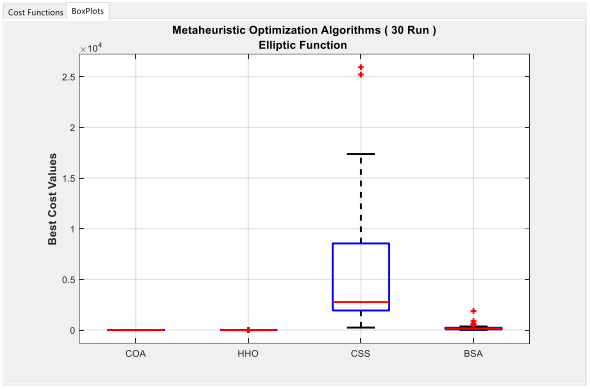
	Boyut Sayısı	**Aralık Değerleri**	***fmin* Değeri**	
Elliptic Function,	30, 50, 100	$(-100, 100)^n$	0	$f_1(x)=\sum_{i=1}^{d}\left(10^6\right)^{\frac{i-1}{d-1}}\times x_i^2$
Non-Continuous Rastrigin Function		$(-5.12, 5.12)^n$	0	$f_2(x)=\sum_{i=1}^{d}\left[x_i^2-10\cos(2\pi x_i)+10\right],$ $x_i=\begin{cases} eğer\ \lvert x_i\rvert\le 0.5 \quad ise \quad x_i \\ eğer\ \lvert x_i\rvert> 0.5 \quad ise \quad round(2\cdot x_i)/2 \end{cases}$
Alpine Function		$(-10, 10)^n$	0	$f_3(x)=\sum_{i=1}^{d}\lvert x_i\sin(x_i)+0.1x_i\rvert$
Levy Function		$(-10, 10)^n$	0	$f_4(x)=\sin^2(\pi w_1)+\sum_{i=1}^{d-1}(w_i-1)^2\left[1+10\sin^2(\pi w_i+1)\right]$ $+(w_d-1)^2\left[1+\sin^2(2\pi w_d)\right]$ $,\quad w_i=1+\frac{x_i-1}{4}$
Weierstrass Function		$(-0.5, 0.5)^n$	0	$f_5(x)=\sum_{i=1}^{d}\left[\sum_{k=0}^{k_{\max}}a^k\cos\left(2\pi b^k(x_i+0.5)\right)\right]-d\sum_{k=0}^{k_{\max}}a^k\cos\left(2\pi b^k\right)$ $,\ a=0.5, b=3, k_{\max}=20$
Michalewicz Function		$(0, \pi)^n$	0	$f_6(x)=-\sum_{i=1}^{d}\sin(x_i)\cdot\left(\sin\left(\frac{i\cdot x_i^2}{\pi}\right)\right)^{2m},\quad m=10$
Dixon&Price Function		$(-10, 10)^n$	0	$f_7(x)=(x_1-1)^2+\sum_{i=2}^{d}i\cdot\left(2x_i^2-x_{i-1}\right)^2$
Schaffer Function	2	$(-100, 100)^n$	0	$f_8(x,y)=0.5+\frac{\sin^2\left(x^2+y^2\right)^2-0.5}{1+0.001\left(x^2+y^2\right)^2}$
Himmelblau Function	2	$(-5, 5)^n$	0	$f_9(x,y)=\left(x^2+y-11\right)^2+\left(x+y^2-7\right)^2$
Kowalik Function	4	$(-5, 5)^n$	$\approx$ 0.000 3075	$f_{10}(x_1,x_2,x_3,x_4)=\sum_{k=1}^{11}\left[a_k-\frac{x_1b_k\left(b_k+x_2\right)}{b_k\left(b_k+x_3\right)+x_4}\right]^2,$ $a=\begin{bmatrix}0.1957,0.1947,0.1735,0.16,0.0844,\\0.627,0.456,0.342,0.323,0.235,0.246\end{bmatrix},$ $b=[0.25,0.5,1,2,4,6,8,10,12,14,16]$

Tablo 2. Çalışmada kullanılan algoritmalara ait parametreler

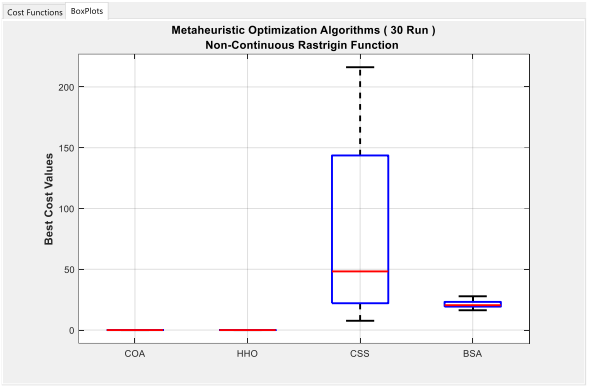
İterasyon sayısı		***PopN***		***FCall***		***Boyut sayısı***	
500		40		15000		2-4-30-50-100	
COA		***HHO***		***CSS***		***BSA***	
σ	3	***E***	[0,2]	***r***	0,01	***mixrate***	2
C1	0,2	***J***	[0,2]	***Start kv***	0.5		
μ	25			***Finish kv***	0,3		
C3	3			***Start ka***	0,5		
				Finish ka	0,5		

4.1. Boyutu 30 Olan Test Fonksiyonları İçin

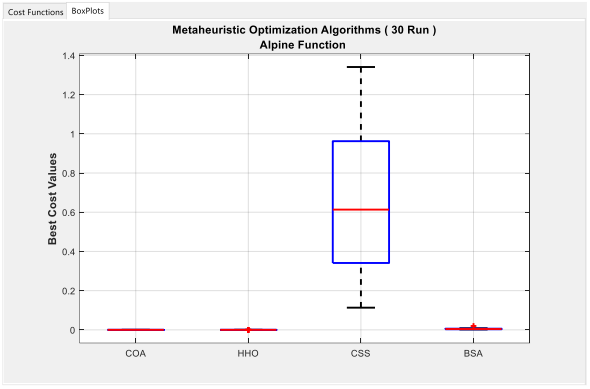
Bu bölümde 30 boyuttaki f_1-f_7 arası test fonksiyonları COA, HHO, CSS ve BSA algoritmalarıyla 30'ar kez çözülmüş ve elde edilen kutu grafikleri Şekil 7-13 arası ve yakınsama eğrileri ise Şekil 14-20 arasında gösterilmiştir. Tüm algoritmalara göre bulunan sonuçlar en kötü, ortalama, en iyi, standart sapma ve süre olarak Tablo 3'de verilmiştir.



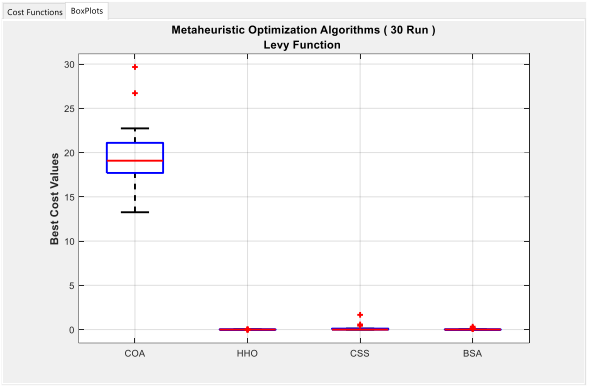
Şekil 7. f1 için kutu grafiği(30-D)



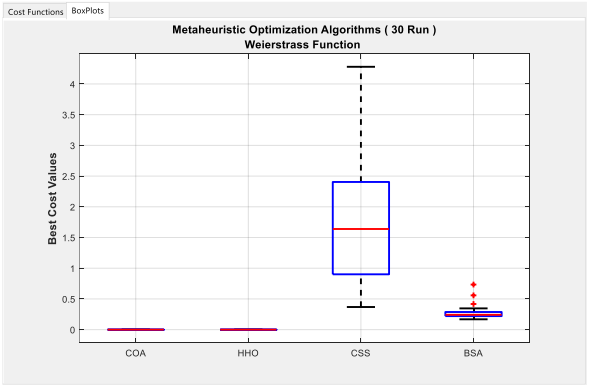
Şekil 8. f2 için kutu grafiği(30-D)



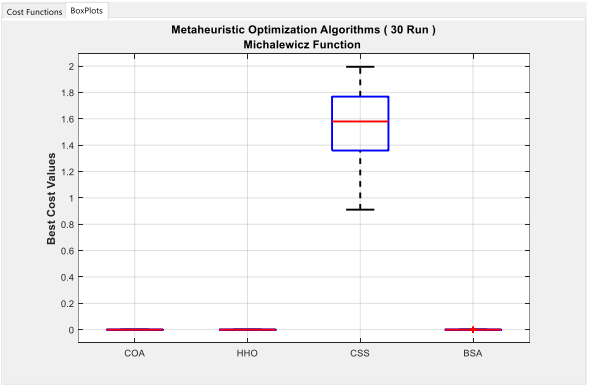
Şekil 9. f3 için kutu grafiği(30-D)



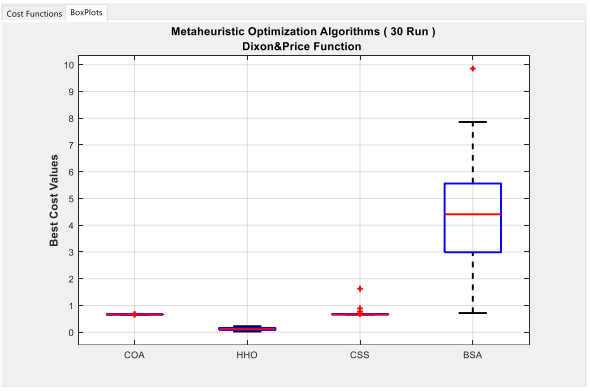
Şekil 10. f4 için kutu grafiği(30-D)



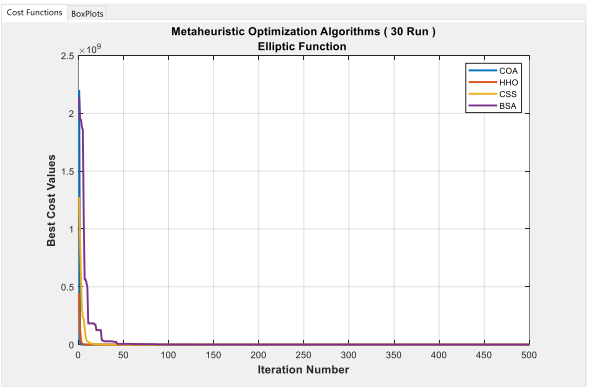
Şekil 11. f5 için kutu grafiği(30-D)



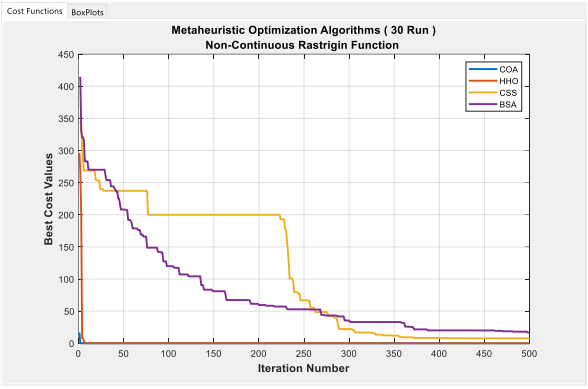
Şekil 12. f6 için kutu grafiği(30-D)



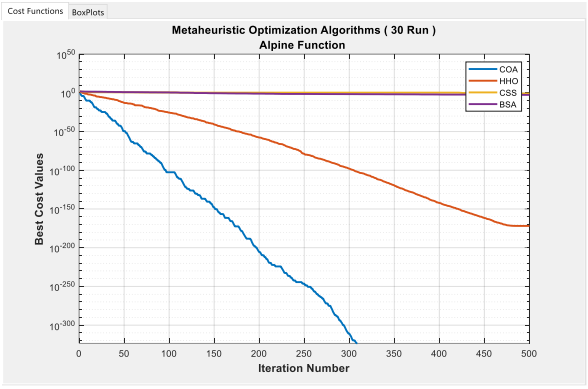
Şekil 13. f7 için kutu grafiği(30-D)



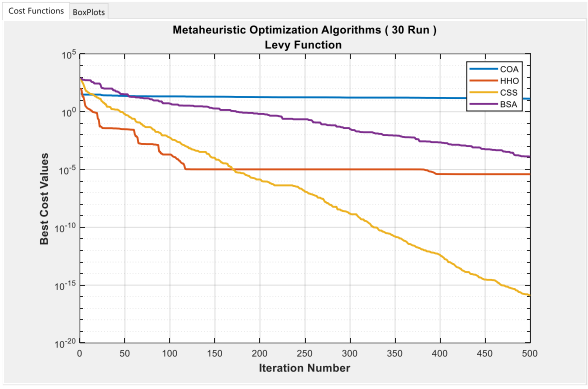
Şekil 14. f1 için yakınsama eğrileri(30-D)



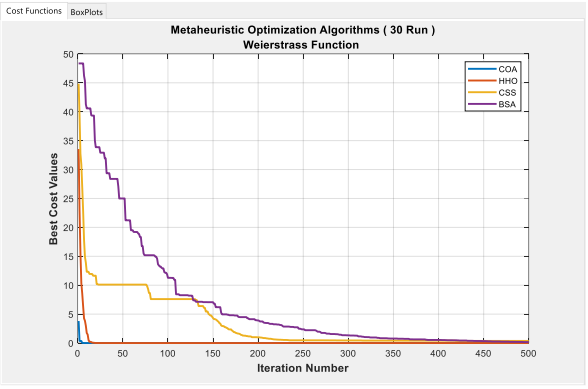
Şekil 15. f2 için yakınsama eğrileri(30-D)



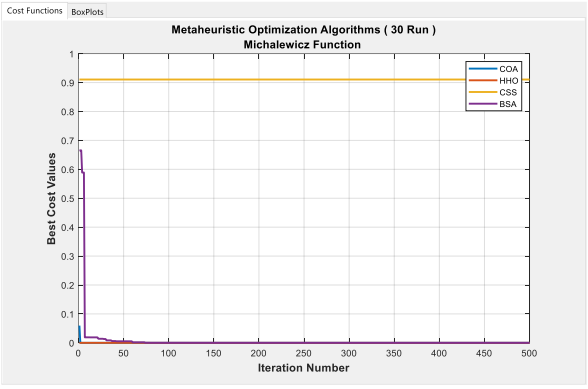
Şekil 16. f3 için yakınsama eğrileri(30-D)



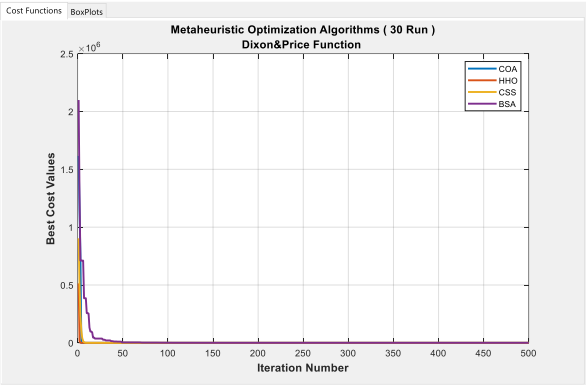
Şekil 17. f4 için yakınsama eğrileri(30-D)



Şekil 18. f5 için yakınsama eğrileri(30-D)



Şekil 19. f6 için yakınsama eğrileri(30-D)



Şekil 20. f7 için yakınsama eğrileri(30-D)

Şekil 7-20 arası gösterilen 30 boyuttaki f_1-f_7 arası test fonksiyonlarının kutu grafikleri ve yakınsama eğrilerini birlikte inceleyerek aşağıdaki değerlendirmeleri yapabiliriz. f_1 ve f_2 test fonksiyonları için Şekil 7-8'de COA ve HHO algoritmalarının diğerlerine göre iyi bir dağılımlara sahip oldukları ve Şekil 14-15'de ise COA'nın HHO'ya göre daha hızlı bir şekilde yakınsayarak en iyi performansları gösterdiği görülmektedir. f_3 test fonksiyonu için Şekil 9'da COA, HHO ve BSA algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 16'da ise COA'nın HHO'ya göre daha iyi performans gösterdiği söylenebilir. f_4 test fonksiyonu için Şekil 10'da HHO algoritmasının en iyi dağılıma sahip olduğu ve Şekil 17'de ise CSS algoritmasının HHO, BSA ve COA'ya göre daha iyi performans gösterdiği ifade edilebilir. f_5 test fonksiyonu için Şekil 11'de COA ve HHO algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 18'de ise COA algoritmasının en iyi performansı göstererek en iyi yakınsamaya sahip olduğu görülmektedir. f_6 test fonksiyonu için Şekil 12'de COA, HHO ve BSA algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 19'da ise HHO algoritmasının COA'ya göre en az iterasyonla yakınsadığı söylenebilir. f_7 test fonksiyonu için Şekil 13'te COA, HHO ve CSS algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 20'de ise HHO algoritmasının COA'ya göre daha hızlı yakınsadığı ifade edilebilir.

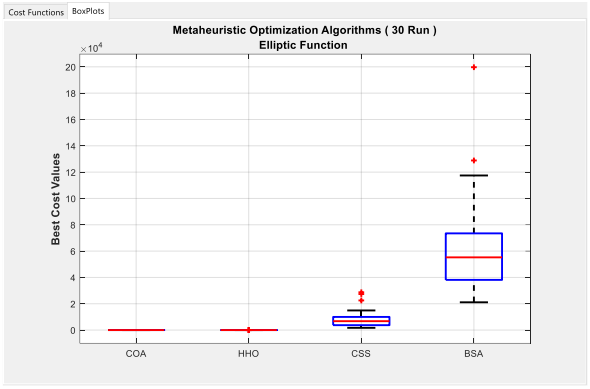
Tablo 3. Algoritmaların 30 Boyutlu test fonksiyonlarına uygulanması durumunda sonuçları (30 bağımsız çalışma)

		COA	*HHO*	*CSS*	*BSA*
f_1	EnKötü	0.000000000e+00	3.70795912e-272	2.593443940e+04	1.884212242e+03
	Ortalama	0.000000000e+00	1.23598743e-273	6.016533696e+03	2.545157701e+02
	EnİyI	0.000000000e+00	5.15654145e-298	2.441273321e+02	1.773107676e+01
	StdSapma	0.000000000e+00	0.000000000e+00	6.840519185e+03	3.622092889e+02
	Süre (s)	0.256895	0.211738	0.472049	0.158166
f_2	EnKötü	0.000000000e+00	0.000000000e+00	2.162244480e+02	2.779835658e+01
	Ortalama	0.000000000e+00	0.000000000e+00	7.749024028e+01	2.089370677e+01
	EnİyI	0.000000000e+00	0.000000000e+00	7.640420598e+00	1.630163451e+01
	StdSapma	0.000000000e+00	0.000000000e+00	6.749270924e+01	2.656748937e+00
	Süre (s)	0.0940452	0.167277	0.475873	0.105965
f_3	EnKötü	0.000000000e+00	5.14554882e-157	1.340737322e+00	1.960382296e-02
	Ortalama	0.000000000e+00	3.25080897e-158	6.553980269e-01	5.083291179e-03
	EnİyI	0.000000000e+00	1.31674151e-172	1.131546252e-01	2.038355882e-03
	StdSapma	0.000000000e+00	9.88896553e-158	3.244607945e-01	3.216886321e-03
	Süre (s)	0.15302	0.0967534	0.373527	0.113696
f_4	EnKötü	2.966732639e+01	7.149647793e-03	1.664973966e+00	3.247722878e-01
	Ortalama	1.927885845e+01	8.214041140e-04	1.258813204e-01	3.014839083e-02
	EnİyI	1.325766679e+01	3.956924182e-06	1.211753808e-16	1.329231841e-04
	StdSapma	3.307148960e+00	1.584175480e-03	3.130086335e-01	6.827050649e-02
	Süre (s)	0.126064	0.230745	0.462959	0.0940044
f_5	EnKötü	0.000000000e+00	0.000000000e+00	4.279579138e+00	7.339623406e-01
	Ortalama	0.000000000e+00	0.000000000e+00	1.751326142e+00	2.783378629e-01
	EnİyI	0.000000000e+00	0.000000000e+00	3.684543025e-01	1.684119921e-01
	StdSapma	0.000000000e+00	0.000000000e+00	9.606739882e-01	1.129142690e-01
	Süre (s)	3.19492	3.53288	2.09585	1.81953
f_6	EnKötü	0.000000000e+00	0.000000000e+00	1.994312920e+00	2.492185246e-08
	Ortalama	0.000000000e+00	0.000000000e+00	1.548527086e+00	8.307286831e-10
	EnİyI	0.000000000e+00	0.000000000e+00	9.106032516e-01	1.434947756e-23
	StdSapma	0.000000000e+00	0.000000000e+00	2.966525027e-01	4.473609377e-09
	Süre (s)	0.256032	0.376848	0.489844	0.197483
f_7	EnKötü	6.666723995e-01	2.250845284e-01	1.627529292e+00	9.855168948e+00
	Ortalama	6.666671711e-01	1.154940209e-01	7.133118136e-01	4.405471984e+00
	EnİyI	6.666666803e-01	2.882321837e-02	6.666669529e-01	7.178599397e-01
	StdSapma	1.067304732e-06	4.662332810e-02	1.752551541e-01	2.001336416e+00
	Süre (s)	0.0593848	0.135372	0.345279	0.0564814

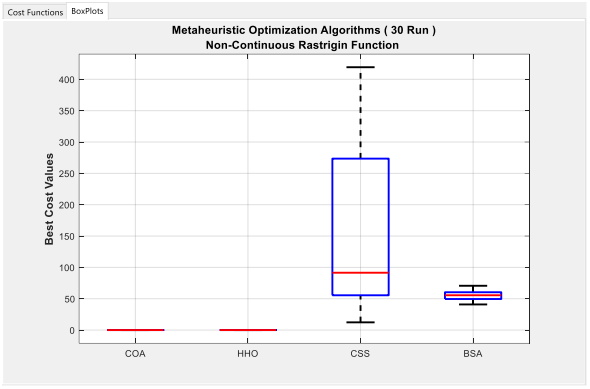
Tablo 3 incelendiğinde COA algoritmasının f_1, f_2, f_3, f_5, *ve* f_6'da en iyi çözüm noktalarında, diğer fonksiyonlarda ise HHO ve CSS algoritmalarının en iyi değerleri yakaladıkları görülmektedir. Ortalama süre açısından ise COA algoritmasının en iyi sürede f_2'de olduğu görülmektedir. HHO algoritmasının en iyi değeri yakaladığı durum f_4 için ortalama değerler ve standart sapmalar açısından karşılaştırıldığında ise, COA algoritmasından daha iyi değerlere sahip olduğu görülmektedir. CSS algoritmalarının en iyi değeri yakaladığı durum f_7 için ortalama değerler açısından karşılaştırıldığında COA algoritmasından daha iyi değerlere sahip olduğu ve standart sapmalar açısından karşılaştırıldığında ise, COA algoritmasından daha kötü değerlere sahip olduğu görülmektedir.

4.2. Boyutu 50 Olan Test Fonksiyonları İçin

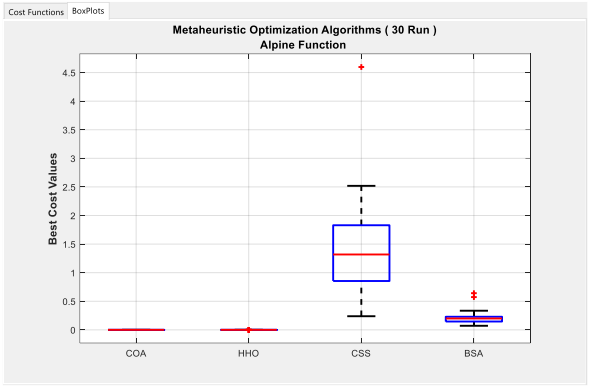
Bu bölümde 50 boyuttaki f_1-f_7 arası test fonksiyonları COA, HHO, CSS ve BSA algoritmalarıyla 30'ar kez çözülmüş ve elde edilen kutu grafikleri Şekil 21-27 arası ve yakınsama eğrileri ise Şekil 28-34 arasında gösterilmiştir. Tüm algoritmalara göre bulunan sonuçlar en kötü, ortalama, en iyi, standart sapma ve süre olarak Tablo 4'te verilmiştir.



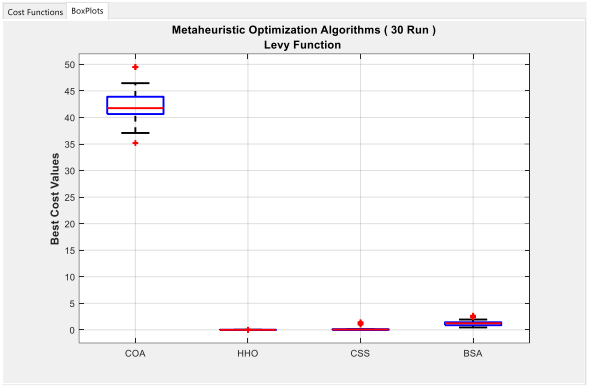
Şekil 21. f1 için kutu grafiği(50-D)



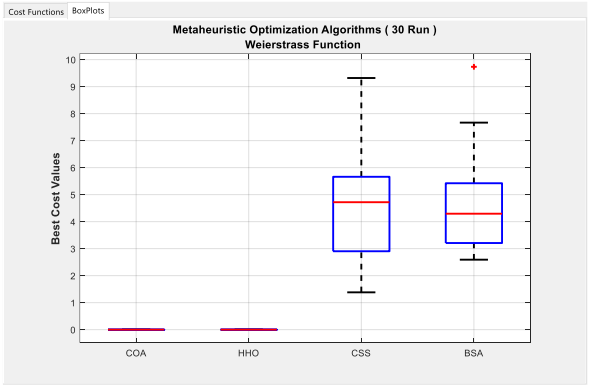
Şekil 22. f2 için kutu grafiği(50-D)



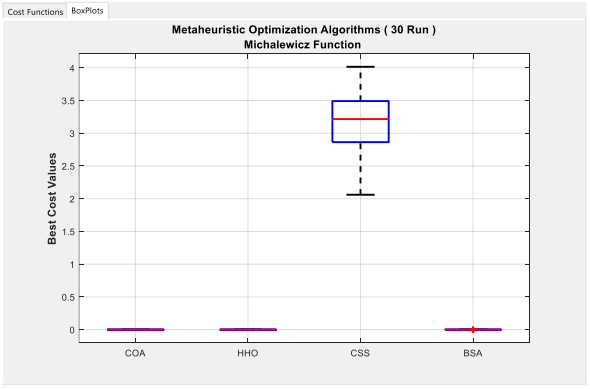
Şekil 23. f3 için kutu grafiği(50-D)



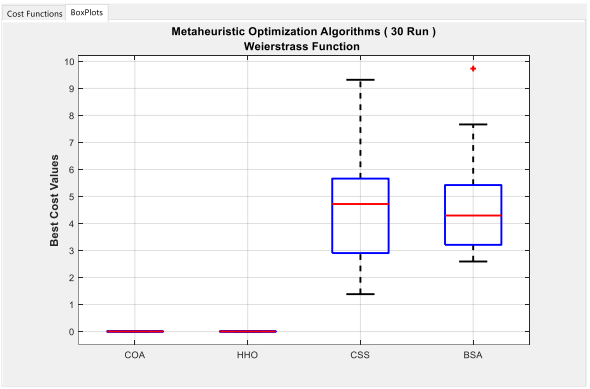
Şekil 24. f4 için kutu grafiği(50-D)



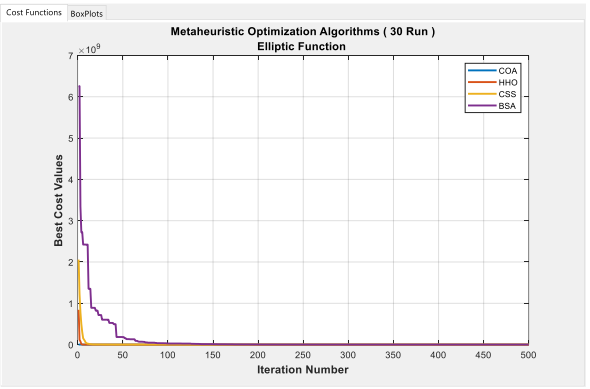
Şekil 25. f5 için kutu grafiği(50-D)



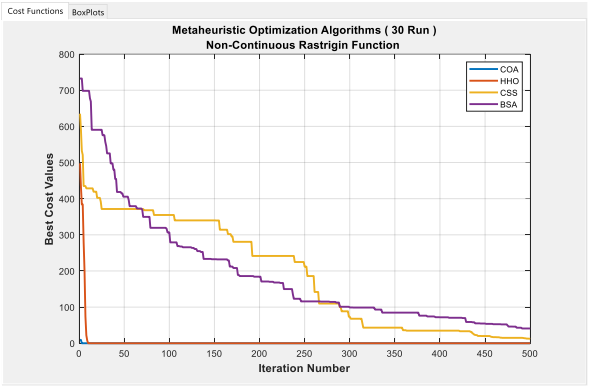
Şekil 26. f6 için kutu grafiği(50-D)



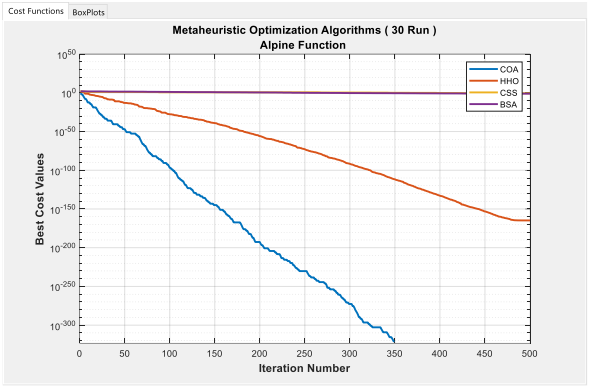
Şekil 27. f7 için kutu grafiği(50-D)



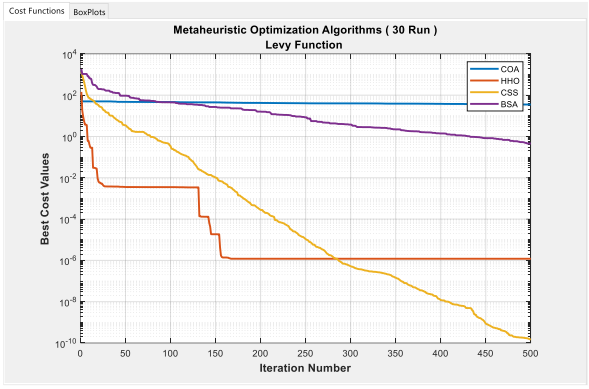
Şekil 28. f1 için yakınsama eğrileri(50-D)



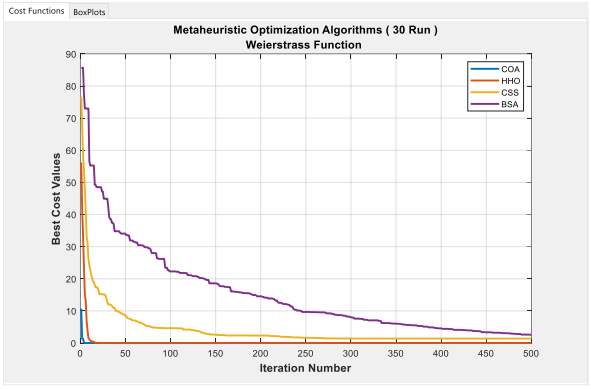
Şekil 29. f2 için yakınsama eğrileri(50-D)



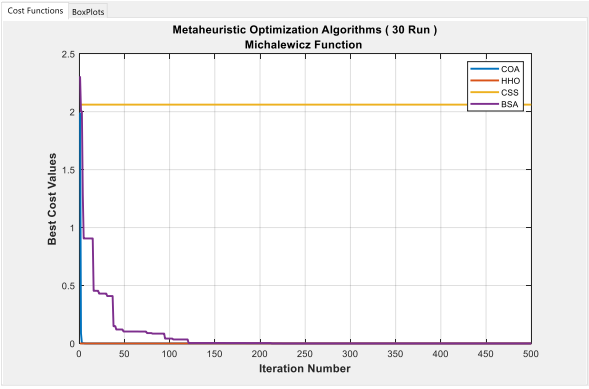
Şekil 30. f3 için yakınsama eğrileri(50-D)



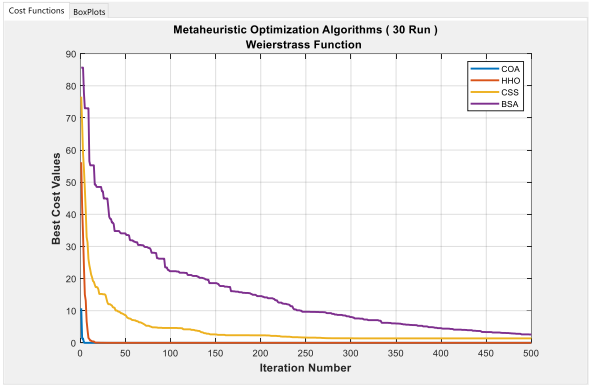
Şekil 31. f4 için yakınsama eğrileri(50-D)



Şekil 32. f5 için yakınsama eğrileri(50-D)



Şekil 33. f6 için yakınsama eğrileri(50-D)



Şekil 34. f7 için yakınsama eğrileri(50-D)

Şekil 21-34 arası gösterilen 50 boyuttaki f_1-f_7 arası test fonksiyonlarının kutu grafikleri ve yakınsama eğrilerini birlikte inceleyerek şu değerlendirmeleri yapabiliriz. f_1 ve f_2 test fonksiyonları için Şekil 21-22'de COA ve HHO algoritmalarının diğerlerine göre iyi bir dağılımlara sahip oldukları ve Şekil 28-29'da ise COA'nın HHO'ya göre daha hızlı bir şekilde yakınsayarak en iyi performansları gösterdiği görülmektedir. f_3 test fonksiyonu için Şekil 23'te COA ve HHO algoritmalarının iyi bir dağılımlara sahip oldukları ve Şekil 30'da ise COA'nın HHO'ya göre daha iyi performans gösterdiği söylenebilir. f_4 test fonksiyonu için Şekil 24'de HHO algoritmasının en iyi dağılıma sahip olduğu ve Şekil 31'de ise CSS algoritmasının HHO, BSA ve COA'ya göre daha iyi performans gösterdiği ifade edilebilir. f_5 test fonksiyonu için Şekil 25'de COA ve HHO algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 32'de COA algoritmasının en iyi performansı göstererek en iyi yakınsamaya sahip olduğu görülmektedir. f_6 test fonksiyonu için Şekil 26'da COA, HHO ve BSA algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 33'te ise HHO algoritmasının COA'ya göre daha iyi yakınsadığı söylenebilir. f_7 test fonksiyonu için Şekil 27'de COA ile HHO algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 34'te ise COA algoritmasının HHO'ya göre daha hızlı yakınsadığı ifade edilebilir.

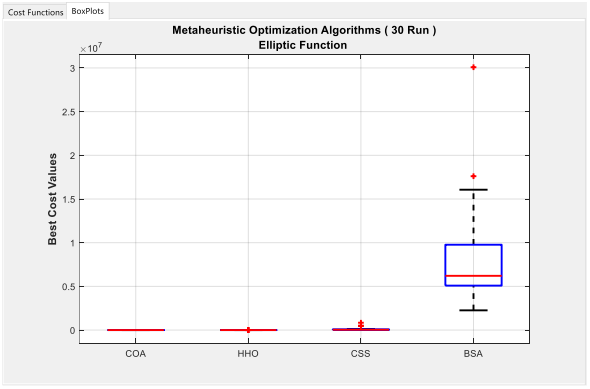
Tablo 4. Algoritmaların 50 Boyutlu test fonksiyonlarına uygulanması durumunda sonuçları (30 bağımsız çalışma)

		COA	*HHO*	*CSS*	*BSA*
f_1	EnKötü	0.000000000e+00	6.41821678e-277	2.871426770e+04	1.996977551e+05
	Ortalama	0.000000000e+00	2.20543306e-278	8.524039799e+03	6.290995202e+04
	Enİyi	0.000000000e+00	2.17773195e-296	1.697173203e+03	2.110301964e+04
	StdSapma	0.000000000e+00	0.000000000e+00	6.795176549e+03	3.691009619e+04
	Süre (s)	0.471005	0.351729	0.637389	0.267548
f_2	EnKötü	0.000000000e+00	0.000000000e+00	4.193025502e+02	7.066658934e+01
	Ortalama	0.000000000e+00	0.000000000e+00	1.573648572e+02	5.548026066e+01
	Enİyi	0.000000000e+00	0.000000000e+00	1.232521178e+01	4.088485310e+01
	StdSapma	0.000000000e+00	0.000000000e+00	1.351401446e+02	7.210412241e+00
	Süre (s)	0.182929	0.438639	0.778613	0.197815
f_3	EnKötü	0.000000000e+00	2.58990167e-156	4.597241239e+00	6.412854319e-01
	Ortalama	0.000000000e+00	2.74891215e-157	1.392081202e+00	2.096120013e-01
	Enİyi	0.000000000e+00	1.71220704e-165	2.383582911e-01	7.118716159e-02
	StdSapma	0.000000000e+00	5.88289395e-157	8.580532196e-01	1.223659873e-01
	Süre (s)	0.206014	0.119756	0.526845	0.131693
f_4	EnKötü	4.949886465e+01	8.679383717e-03	1.441088269e+00	2.668654382e+00
	Ortalama	4.221783959e+01	8.525003795e-04	1.655641668e-01	1.282638371e+00
	Enİyi	3.518430004e+01	1.199906907e-06	1.475388260e-10	4.569776301e-01
	StdSapma	3.207264279e+00	1.792242215e-03	3.586133844e-01	5.898837705e-01
	Süre (s)	0.157499	0.384402	0.538354	0.157207
f_5	EnKötü	0.000000000e+00	0.000000000e+00	9.318741365e+00	9.734084080e+00
	Ortalama	0.000000000e+00	0.000000000e+00	4.578699805e+00	4.603357036e+00
	Enİyi	0.000000000e+00	0.000000000e+00	1.381571537e+00	2.590997463e+00
	StdSapma	0.000000000e+00	0.000000000e+00	1.722825154e+00	1.617311691e+00
	Süre (s)	5.16284	6.3429	3.4381	2.81924
f_6	EnKötü	0.000000000e+00	0.000000000e+00	4.015088642e+00	3.324551946e-05
	Ortalama	0.000000000e+00	0.000000000e+00	3.154592879e+00	4.914669733e-06
	Enİyi	0.000000000e+00	0.000000000e+00	2.060935886e+00	1.497777627e-16
	StdSapma	0.000000000e+00	0.000000000e+00	4.645106288e-01	8.646844487e-06
	Süre (s)	0.436154	1.04624	0.745826	0.327373
f_7	EnKötü	6.666934079e-01	2.497433539e-01	5.248177066e+00	1.326534283e+02
	Ortalama	6.666766758e-01	1.629861398e-01	8.463822177e-01	4.607752485e+01
	Enİyi	6.666674653e-01	7.391894133e-02	6.666703273e-01	2.138019777e+01
	StdSapma	7.824743661e-06	4.287515677e-02	8.193092468e-01	2.134020147e+01
	Süre (s)	0.0834142	0.242556	0.517351	0.078468

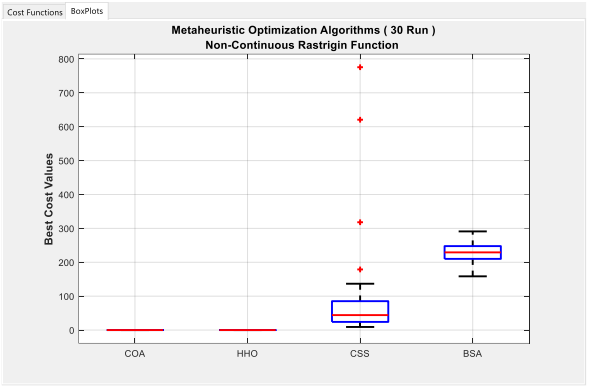
Tablo 4 incelendiğinde COA algoritmasının f_1, f_2, f_3, f_5 *ve* f_6'da en iyi çözüm noktalarında, diğer fonksiyonlarda ise HHO ve CSS algoritmalarının en iyi değerleri yakaladıkları görülmektedir. Ortalama süre açısından da COA algoritmasının en iyi sürede f_2'de olduğu görülmektedir. HHO algoritmasının en iyi değeri yakaladığı durum f_4 için ortalama değerler ve standart sapmalar açısından karşılaştırıldığında ise, COA algoritmasından daha iyi değerlere sahip olduğu görülmektedir. CSS algoritmalarının en iyi değeri yakaladığı durum f_7 için ortalama değerler açısından karşılaştırıldığında COA algoritmasından daha iyi değerlere sahip olduğu ve standart sapmalar açısından karşılaştırıldığında ise, COA algoritmasından daha kötü değerlere sahip olduğu görülmektedir.

4.3. Boyutu 100 Olan Test Fonksiyonları İçin

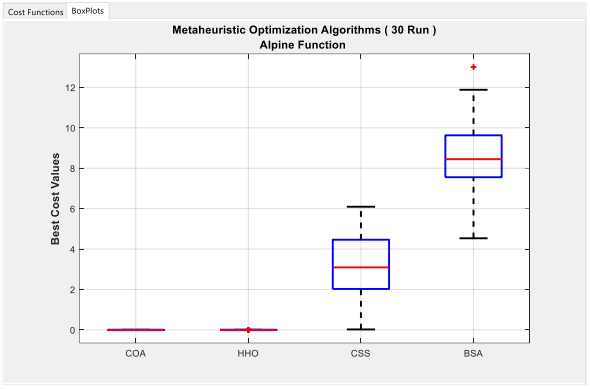
Bu bölümde 100 boyuttaki f_1-f_7 arası test fonksiyonları COA, HHO, CSS ve BSA algoritmalarıyla 30'ar kez çözülmüş ve elde edilen kutu grafikleri Şekil 35-41 arası ve yakınsama eğrileri ise Şekil 42-48 arasında gösterilmiştir. Tüm algoritmalara göre bulunan sonuçlar en kötü, ortalama, en iyi, standart sapma ve süre olarak Tablo 5'te verilmiştir.



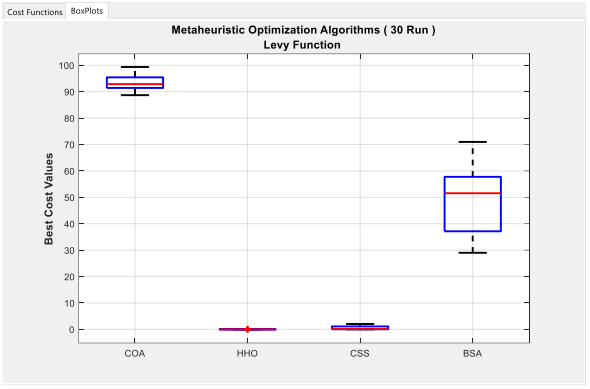
Şekil 35. f1 için kutu grafiği(100-D)



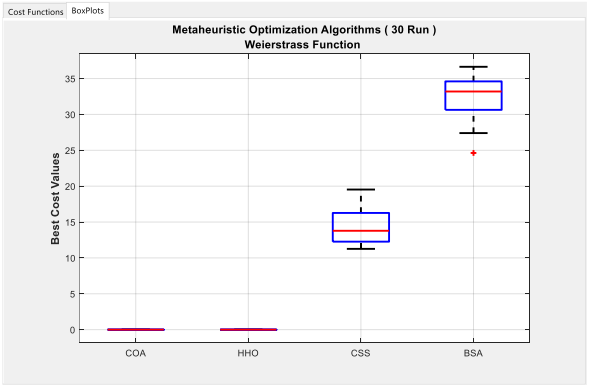
Şekil 36. f2 için kutu grafiği(100-D)



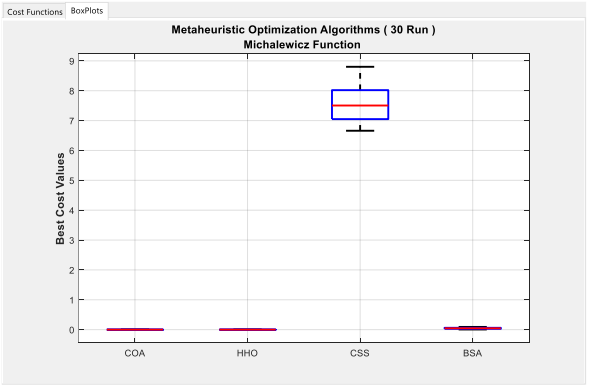
Şekil 37. f3 için kutu grafiği(100-D)



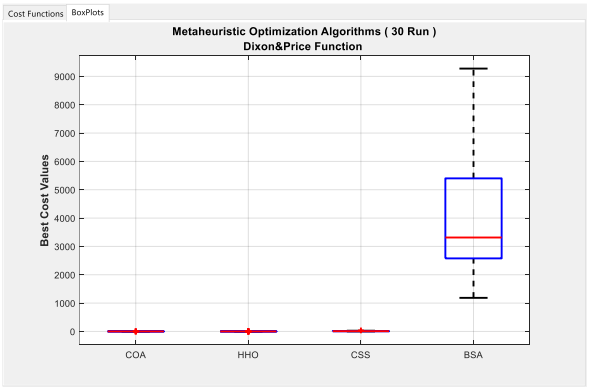
Şekil 38. f4 için kutu grafiği(100-D)



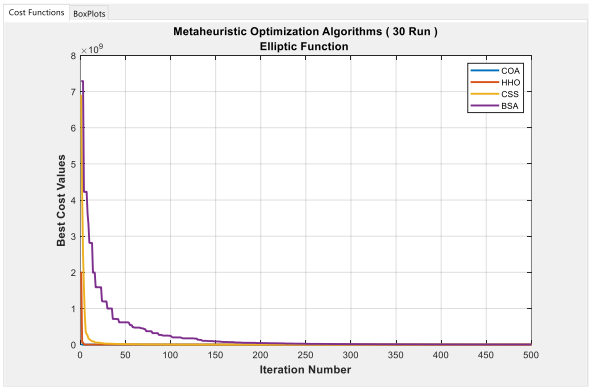
Şekil 39. f5 için kutu grafiği(100-D)



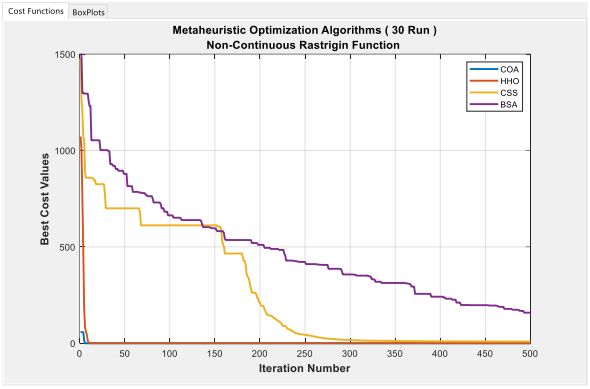
Şekil 40. f6 için kutu grafiği(100-D)



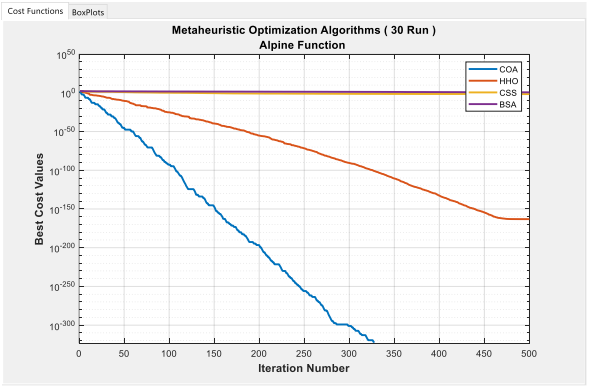
Şekil 41. f7 için kutu grafiği(100-D)



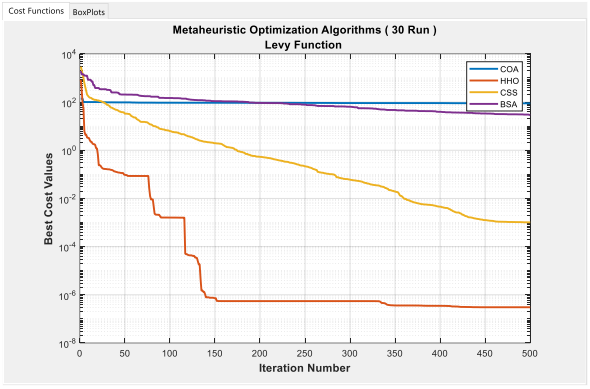
Şekil 42. f1 için yakınsama eğrileri(100-D)



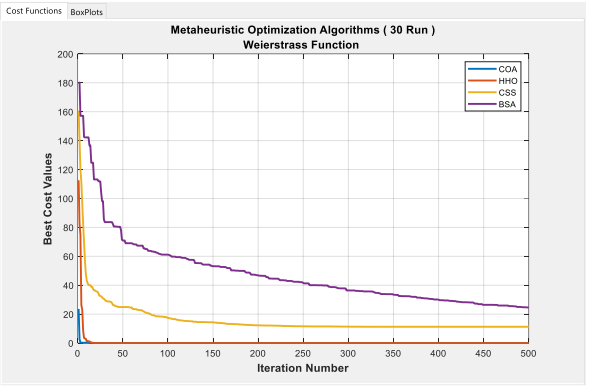
Şekil 43. f2 için yakınsama eğrileri(100-D)



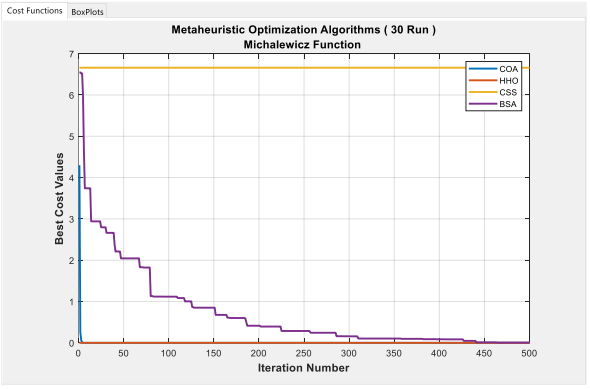
Şekil 44. f3 için yakınsama eğrileri(100-D)



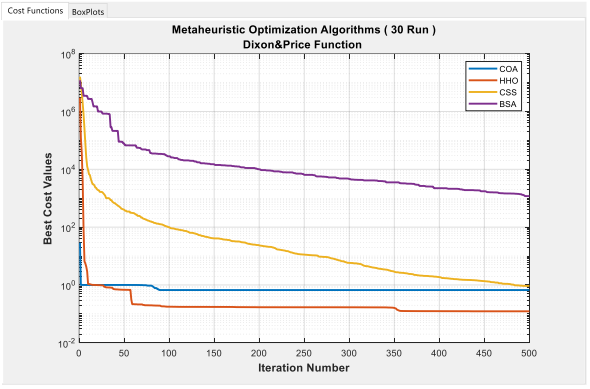
Şekil 45. f4 için yakınsama eğrileri(100-D)



Şekil 46. f5 için yakınsama eğrileri(100-D)



Şekil 47. f6 için yakınsama eğrileri(100-D)



Şekil 48. f7 için yakınsama eğrileri(100-D)

Şekil 35-48 arası gösterilen 100 boyuttaki f_1-f_7 arası test fonksiyonlarının kutu grafikleri ve yakınsama eğrilerini birlikte inceleyerek aşağıdaki değerlendirmeleri yapabiliriz. f_1 ve f_2 test fonksiyonları için Şekil 35-36'da COA ve HHO algoritmalarının diğerlerine göre iyi bir dağılımlara sahip oldukları ve Şekil 42-43'te ise COA'nın HHO'ya göre daha hızlı bir şekilde yakınsayarak en iyi performansları gösterdiği görülmektedir. f_3 test fonksiyonu için Şekil 37'de COA ve HHO algoritmalarının iyi bir dağılımlara sahip oldukları ve Şekil 44'te ise COA'nın diğer algoritmalara göre daha iyi performans gösterdiği söylenebilir. f_4 test fonksiyonu için Şekil 38'de HHO ve CSS algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 45'de ise HHO, CSS ve BSA algoritmalarının COA'ya göre daha iyi performans gösterdiği ifade edilebilir. f_5 test fonksiyonu için Şekil 39'da COA ve HHO algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 46'da ise COA'nın diğer algoritmalara göre daha iyi performans gösterdiği söylenebilir. f_6 test fonksiyonu için Şekil 40'da HHO, BSA ile COA algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 47'de ise HHO algoritmasının COA'ya göre daha iyi performans gösterdiği ifade edilebilir. f_7 test fonksiyonu için Şekil 41'de COA, HHO ve CSS algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 48'de ise HHO algoritmasının COA'ya göre daha hızlı yakınsadığı ifade edilebilir.

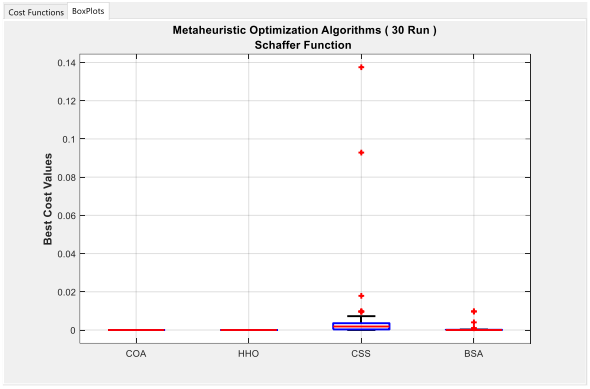
Tablo 5. Algoritmaların 100 Boyutlu test fonksiyonlarına uygulanması durumunda sonuçları (30 bağımsız çalışma)

		COA	*HHO*	*CSSJ*	*BSA*
f_1	EnKötü	0.000000000e+00	4.94197136e-276	8.392134107e+05	3.008172789e+07
	Ortalama	0.000000000e+00	1.70090182e-277	1.017145069e+05	8.117098571e+06
	EnİYi	0.000000000e+00	2.27855811e-295	8.149396871e+03	2.257152310e+06
	StdSapma	0.000000000e+00	0.000000000e+00	1.773132206e+05	5.553688722e+06
	Süre (s)	0.903167	0.607552	0.903743	0.447395
f_2	EnKötü	0.000000000e+00	0.000000000e+00	7.754217525e+02	2.910749230e+02
	Ortalama	0.000000000e+00	0.000000000e+00	1.035094238e+02	2.293255475e+02
	EnİYi	0.000000000e+00	0.000000000e+00	9.142990366e+00	1.585523854e+02
	StdSapma	0.000000000e+00	0.000000000e+00	1.714588525e+02	3.282452937e+01
	Süre (s)	0.195455	0.348959	0.619033	0.198577
f_3	EnKötü	0.000000000e+00	1.27449386e-154	6.093792619e+00	1.301058367e+01
	Ortalama	0.000000000e+00	7.49256779e-156	3.206623703e+00	8.549302892e+00
	EnİYi	0.000000000e+00	7.94069719e-164	2.101854071e-02	4.532600510e+00
	StdSapma	0.000000000e+00	2.71506276e-155	1.670842189e+00	1.811710678e+00
	Süre (s)	0.3144	0.12431	0.565801	0.121941
f_4	EnKötü	9.933918986e+01	1.654379434e-02	2.015888167e+00	7.097438424e+01
	Ortalama	9.350822820e+01	1.220085645e-03	5.124034157e-01	4.903051232e+01
	EnİYi	8.867370472e+01	3.046131352e-07	1.018482902e-03	2.898540536e+01
	StdSapma	2.944243084e+00	2.940982370e-03	6.374421062e-01	1.207821902e+01
	Süre (s)	0.215774	0.41846	0.624653	0.17086
f_5	EnKötü	0.000000000e+00	0.000000000e+00	1.952937480e+01	3.663805112e+01
	Ortalama	0.000000000e+00	0.000000000e+00	1.428026756e+01	3.255078018e+01
	EnİYi	0.000000000e+00	0.000000000e+00	1.125964899e+01	2.462647426e+01
	StdSapma	0.000000000e+00	0.000000000e+00	2.415380110e+00	2.888057420e+00
	Süre (s)	9.12338	10.1082	5.44946	4.96749
f_6	EnKötü	0.000000000e+00	0.000000000e+00	8.801297790e+00	9.096605556e-02
	Ortalama	0.000000000e+00	0.000000000e+00	7.552437615e+00	4.562120093e-02
	EnİYi	0.000000000e+00	0.000000000e+00	6.659663915e+00	5.905831101e-03
	StdSapma	0.000000000e+00	0.000000000e+00	5.942220006e-01	2.107831378e-02
	Süre (s)	0.795862	1.12568	1.01241	0.581091
f_7	EnKötü	6.670067579e-01	2.508023605e-01	3.567204171e+01	9.275321920e+03
	Ortalama	6.667310859e-01	1.803532231e-01	1.047601157e+01	4.037083487e+03
	EnİYi	6.666766668e-01	1.227066305e-01	9.270521615e-01	1.184919624e+03
	StdSapma	6.842054084e-05	3.725655588e-02	8.010874527e+00	2.091156085e+03
	Süre (s)	0.152208	0.289212	0.651036	0.116422

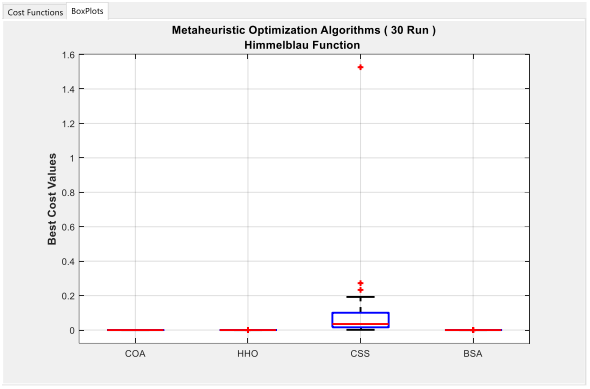
Tablo 5 incelendiğinde COA algoritmasının f_1, f_2, f_3, f_5 *ve* f_6'da en iyi çözüm noktalarında, diğer fonksiyonlarda ise HHO algoritmasının en iyi değeri yakaladıkları görülmektedir. Ortalama süre açısından da COA algoritmasının en iyi sürede f_2'de olduğu görülmektedir. HHO algoritmasının en iyi değeri yakaladığı durum f_4'de ortalama değerler ve standart sapmalar açısından karşılaştırıldığında ise, COA algoritmasından daha iyi değerlere sahip olduğu görülmektedir. HHO algoritmasının en iyi değeri yakaladığı durum f_7'de ortalama değerler açısından karşılaştırıldığında COA algoritmasından daha iyi değerlere sahip olduğu ve standart sapmalar açısından karşılaştırıldığında ise, COA algoritmasından daha kötü değerlere sahip olduğu görülmektedir.

4.4. Sabit Boyutlu Test Fonksiyonları İçin

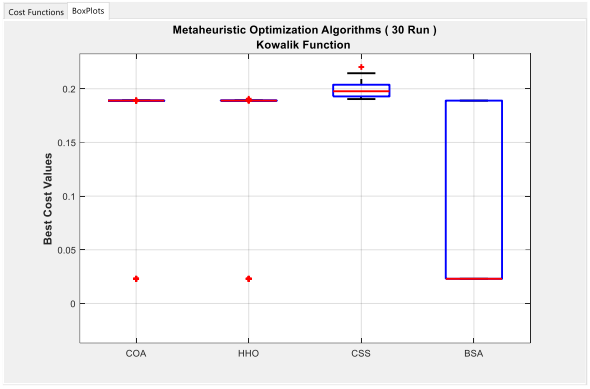
Bu bölümde 2 boyuttaki f_8 ve f_9, 4 boyuttaki f_{10} test fonksiyonları COA, HHO, CSS ve BSA algoritmalarıyla 30'ar kez çözülmüş ve elde edilen kutu grafikleri Şekil 49-51 arası ve yakınsama eğrileri ise Şekil 52-54 arasında gösterilmiştir. Tüm algoritmalara göre bulunan sonuçlar en kötü, ortalama, en iyi, standart sapma ve süre olarak Tablo 6'da verilmiştir.



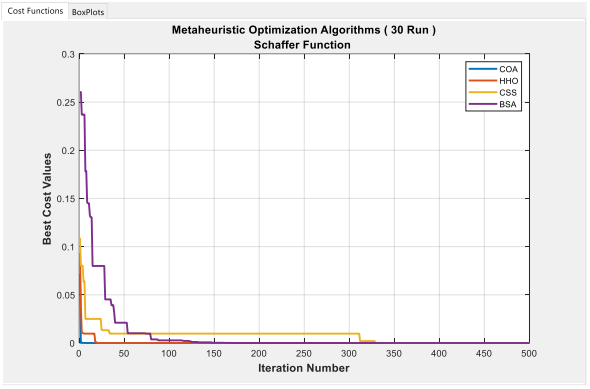
Şekil 49.1 f8 için kutu grafiği(2-D)



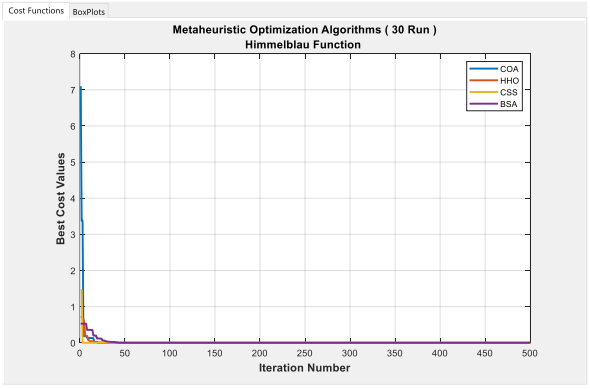
Şekil 50. f9 için kutu grafiği(2-D)



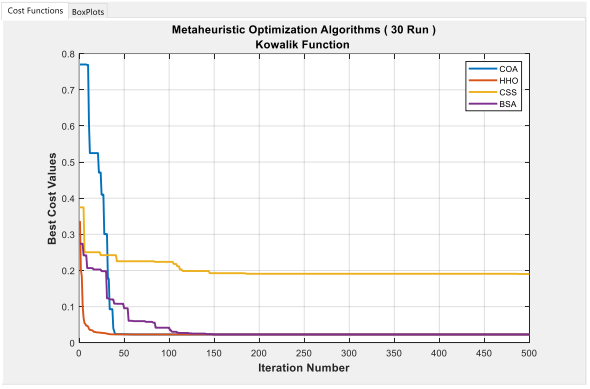
Şekil 51. f10 için kutu grafiği(4-D)



Şekil 52. f8 için yakınsama eğrileri(2-D)



Şekil 53. f9 için yakınsama eğrileri(2-D)



Şekil 54. f10 için yakınsama eğrileri(4-D)

Şekil 49-54 arası gösterilen sabit boyuttaki f_8-f_{10} arası test fonksiyonlarının kutu grafikleri ve yakınsama eğrilerini birlikte inceleyerek aşağıdaki değerlendirmeleri yapabiliriz. f_8 test fonksiyonu için Şekil 49'da COA ve HHO algoritmalarının diğerlerine göre iyi bir dağılımlara sahip oldukları ve Şekil 52'de ise COA'nın HHO'ya göre daha hızlı bir şekilde aynı değere yakınsadıkları görülmektedir. f_9 test fonksiyonu için Şekil 50'de COA, HHO ve BSA algoritmalarının iyi bir dağılımlara sahip oldukları ve Şekil 53'te ise CSS ve HHO algoritmalarının COA'ya göre daha hızlı ve fakat tüm algoritmaların aynı değere yakınsadıkları söylenebilir. f_{10} test fonksiyonu için Şekil 51'de HHO ve COA algoritmalarının diğerlerinden daha iyi bir dağılıma sahip olduğu ve Şekil 54'te ise HHO algoritmasının COA'ya göre daha hızlı yakınsama değerine sahip olduğu ifade edilebilir.

Tablo 6 incelendiğinde COA algoritmasının f_8, *ve* f_9'da en iyi çözüm noktalarında, diğer fonksiyonda ise BSA algoritmasının en iyi değeri yakaladıkları görülmektedir. Ortalama süre açısından da COA algoritmasının en iyi sürede f_8, f_9 *ve* f_{10}'da olduğu görülmektedir. BSA algoritmasının en iyi değeri yakaladığı durum f_{10} için ortalama değerler açısından karşılaştırıldığında COA algoritmasından daha iyi olduğu ve standart sapmalar açısından karşılaştırıldığında ise COA algoritmasından daha kötü değerlere sahip olduğu görülmektedir.

Tablo 6. Algoritmaların Sabit boyutlu test fonksiyonlarına uygulanması durumunda sonuçları (30 bağımsız çalışma)

		COA	*HHO*	*CSS*	*BSA*
f_8	EnKötü	0.000000000e+00	0.000000000e+00	1.375220729e-01	9.715909878e-03
	Ortalama	0.000000000e+00	0.000000000e+00	1.050293825e-02	8.638555997e-04
	Enİyi	0.000000000e+00	0.000000000e+00	3.300863938e-05	0.000000000e+00
	StdSapma	0.000000000e+00	0.000000000e+00	2.884166951e-02	2.478597898e-03
	Süre (s)	0.0403364	0.0820939	0.261233	0.0408193
f_9	EnKötü	7.888609052e-31	1.577721810e-29	1.526359564e+00	1.959677532e-22
	Ortalama	3.418397256e-31	1.367358902e-30	1.134555325e-01	6.589065130e-24
	Enİyi	0.000000000e+00	0.000000000e+00	1.416855131e-03	0.000000000e+00
	StdSapma	3.909086816e-31	3.161573252e-30	2.717440129e-01	3.516722141e-23
	Süre (s)	0.0238806	0.065922	0.271835	0.0298616
f_{10}	EnKötü	1.889337932e-01	1.906626048e-01	2.202368095e-01	1.889337486e-01
	Ortalama	1.557456736e-01	1.558400003e-01	1.993740795e-01	8.383816546e-02
	Enİyi	2.299335416e-02	2.299335416e-02	1.903879040e-01	2.299335416e-02
	StdSapma	6.637615971e-02	6.642365387e-02	7.703616069e-03	7.996574845e-02
	Süre (s)	0.0297623	0.0764176	0.264134	0.0380855

5. Sonuçlar

Bu çalışmada doğadaki kerevitlerin yiyecek arama, büyük besinleri ayaklarıyla parçalama, yaz tatili için yer arama ve rekabetçi davranışlarından ilham alan, yani doğadan ilham alan yeni bir sürü tabanlı meta-sezgisel algoritma olan COA tanıtılarak çeşitli test fonksiyonları üzerinde uygulanmıştır. Yiyecek arama aşaması ve rekabet aşaması COA'nın kullanım(explotation) aşamasıdır ve yaz tatili aşaması ise COA'nın keşif aşamasıdır.

MA'lar fizik, evrim, sürü ve insan tabanlı olmak üzere dört farklı ana kategoride oldukları için COA ile elde edilen sonuçlar, HHO, CSS ve BSA algoritmalarının sonuçlarıyla karşılaştırılmıştır. Sürü tabanlı algoritmaya örnek olan COA'ya karşılık çalışmada literatürden karşılaştırmak için seçilen algoritmalardan HHO sürü tabanlı, CSS fizik tabanlı ve BSA ise evrim tabanlı algoritmalardır. Algoritma seçim işleminde karşılaştırma için seçilen MA'ların bir adedinin aynı tabanlı gruptan diğer ikisinin ise farklı tabanlı gruptan olmasına özen gösterilmiştir.

Çalışmada biri unimodal, altı adedi multimodal olan 7 adet test fonksiyon 30, 50 ve 100 boyutlu ve üç adedi ise sabit boyutlu fonksiyon olmak üzere toplam on adet test fonksiyonu kullanılmıştır. Seçilen 30 ve 50 boyutlu yedi adet test fonksiyonun çözümlerinde en iyi değerlere sahip algoritmalar sırasıyla COA (1 adet unimodal test fonksiyonu, 4 adet multimodal test fonksiyonu), HHO (1 adet multimodal test fonksiyonu) ve CSS (1 adet multimodal test fonksiyonu) olmuştur. Yedi adet test fonksiyonunun 100 boyutlu olması durumunda ise en iyi değerlere sahip algoritmalar sırasıyla COA (1 adet unimodal test fonksiyonu, 4 adet multimodal test fonksiyonu) ve HHO (2 adet multimodal test fonksiyonu) olmuştur. Sabit boyut için yapılan değerlendirmede en iyi değerlere sahip algoritmalar sırasıyla COA (2 adet multimodal test fonksiyonu) ve BSA (1 adet multimodal test fonksiyonu) olmuştur. Çalışmada tüm kutu grafikleri ve yakınsama eğrileri birlikte incelendiğinde sırasıyla COA ve HHO algoritmalarının en başarılı algoritmalar olduğu görülmektedir.

Bu çalışmada elde edilen sonuçlara göre COA algoritmasının şu avantajlara sahip olduğu gözlemlenmiştir; Farklı boyutlardaki optimizasyon problemlerinde yüksek çözüm doğruluğu göstermektedir. Özellikle düşük ve orta boyutlu test fonksiyonlarında hızlı yakınsama sağlamaktadır. Hem keşif (exploration) hem de sömürü (exploitation) aşamalarında dengeli bir performans sergilemektedir. Parametre ayarlarının sayısının az olması uygulama kolaylığı sağlamaktadır. Algoritmanın bazı dezavantajları da şu şekilde belirtilebilir; Çok yüksek boyutlu problemlerde (özellikle 100 boyut ve üzeri) algoritmanın yakınsama hızı zaman zaman diğer yöntemlere göre daha düşük kalabilmektedir. Yerel minimumlarda takılma riski düşük olmakla birlikte tamamen ortadan kalkmamıştır; bazı fonksiyonlarda erken yakınsama davranışı gözlemlenmiştir. Başlangıç popülasyonu çeşitliliğinin az olması durumunda performans kaybı yaşanabilmektedir. Bu bulgular, COA algoritmasının genel olarak etkili bir optimizasyon yaklaşımı sunduğunu, ancak performansını artırmak için özellikle çok yüksek boyutlu problemlerde adaptif stratejilerle desteklenmesi gerektiğini ortaya koymaktadır.

Bu çalışma, COA algoritmasının çeşitli test fonksiyonları üzerindeki performansını incelemeye yönelik bir ön araştırma niteliğindedir. Gelecekteki çalışmalardan biri, bu algoritmanın elektrik-elektronik mühendisliğinde önemli bir konu olan ekonomik güç dağıtımı problemlerine uygulanmasını içerecektir.

Referanslar

[1] B., Öztürk, L., Uğur, F., Erzincanlı, and Ö. Küçük, "Optimization of Polyethylene Inserts Design Geometry of Total Knee Prosthesis," *International Scientific and Vocational Studies Journal*, vol. 2, no. 2, pp. 31-39, 2018.

[2] İ. B., Koç, A. A., Janadi, and V. Ateş, "Interlock Optimization of an Accelerator using Genetic Algorithm," *International Scientific and Vocational Studies Journal,* vol. 1, no. 1, 30-41, 2017.

[3] Y., Altınok, M., Lüy, N. A., Metin, S., Görgülü Balcı, and F., Acar, "Sustainable Grids: Smart Meter Solutions for Efficient Energy Measurement," *International Scientific and Vocational Studies Journal*, vol. 8, no. 1, pp. 49-64, 2024. DOI: 10.47897/bilmes.1485662.

[4] K., Eryılmaz, and A., Elen, "Changes in Traffic Density and Vehicle Usage Habits During and After the Pandemic in Balıkesir Province," *International Scientific and Vocational Studies Journal*, vol. 8, no. 2, pp. 235-252, 2024. DOI: 10.47897/bilmes.1602255.

[5] S., Özyön, C., Yaşar, and H., Temurtaş, "Kaos Tabanlı Yerçekimsel Arama Algoritmaları (CbGSA-X) için Test Fonksiyonları," *Düzce Üniversitesi Bilim ve Teknoloji Dergisi*, vol. 8, no. 3, pp. 1771-1793, 2020.

[6] F. Cantaş, S. Özyön, and C. Yaşar, "Runge Kutta Optimization for Fixed Size Multimodal Test Functions," *International Scientific and Vocational Studies Journal*, vol. 6, no. 2, pp. 144-155, 2022. DOI: 10.47897/bilmes.1219033.

[7] H., Jia, H., Rao, C., Wen, and S., Mirjalili, "Crayfish Optimization Algorithm," *Artificial Intelligence Review*, vol. 56 (Suppl 2), pp. 1919-1979, 2023. DOI: 10.1007/s10462-023-10567-4.

[8] S., Özyön, C., Yaşar, and H., Temurtaş, "Particle Swarm Optimization Algorithm Applied to Environmental Economic Power Dispatch Problems Consisting of Thermal Units," *In 6th International Advanced Technologies Symposium (IATS 2011),* Electrical & Electronics Technologies Papers, vol. 4, EAE-39, pp. 175-180, Elâzığ, Turkey, May 16-18, 2011.

[9] D., Aydın, G., Yavuz, S., Özyön, C., Yaşar, and T., Stützle, "Artificial Bee Colony Framework to Non-convex Economic Dispatch Problem with Valve Point Effects," *In Genetic and Evolutionary Computation Conference Companion (GECCO'17),* pp. 1311-1318, Berlin, Germany, July 15–17, 2017. ISBN: 978-1-4503-4939-0.

[10] C., Yaşar, and S., Özyön, "A Modified Incremental Gravitational Search Algorithm for Short-term Hydrothermal Scheduling with Variable Head," *Engineering Applications of Artificial Intelligence*, vol. 95, pp. 1-17. article 103845, 2020. DOI: 10.1016/j.engappai.2020.103845.

[11] A., Kaveh, and A., Dadras, "A Novel Meta-heuristic Optimization Algorithm: Thermal Exchange Optimization," *Advances in Engineering Software*, vol. 110, pp. 69-84, 2017. DOI: 10.1016/j.advengsoft.2017.03.014.

[12] S., Özyön, B., Durmuş, G., Kuvat, and G., Özcan, "Yüksek Boyutlu Problemlerin Optimizasyonunda Parametre Seçiminin Genetik Algoritma Performansına Etkileri," *In International Multidisciplinary Congress of Eurasia (IMCOFE'15),* pp. 842-859, Üsküp, Macedonia, September 1-5, 2015. ISBN: 978-9944-0637-1-5.

[13] P., Civicioglu, "Backtracking Search Optimization Algorithm for Numerical Optimization Problems," *Applied Mathematics and Computation*, vol. 219, no. 15, pp. 8121-8144, 2013. DOI: 10.1016/j.amc.2013.02.017.

[14] S., Özyön, Differential Evolution Algorithm with Incremental Social Learning," *Bilecik Şeyh Edebali Üniversitesi Fen Bilimleri Dergisi*, vol. 7, no. 1, pp. 133-163, 2020. DOI: 10.35193/bseufbd.666626.

[15] S., Özyön, C., Yaşar, and H., Temurtaş, "Harmoni Arama Algoritmasının Çevresel Ekonomik Güç Dağıtım Problemlerine Uygulanması," *Çukurova Üniversitesi Mühendislik-Mimarlık Fakültesi Dergisi*, vol. 26, no. 2, pp. 65-76, 2011.

[16] R. V., Rao, V. J., Savsani, and D. P., Vakharia, "Teaching-learning-based Optimization: A Novel Method for Constrained Mechanical Design Optimization Problems," *Computer-Aided Design*, vol. 43, no. 3, pp. 303-315, 2011. DOI: 10.1016/j.cad.2010.12.015.

[17] A. A., Heidari, S., Mirjalili, H., Faris, I., Aljarah, M., Mafarja, and H., Chen, "Harris Hawks Optimization: Algorithm and Applications," *Future Generation Computer Systems*, vol. 97, pp. 849-872, 2019. DOI: 10.1016/j.future.2019.02.028.

[18] O., Akdağ, A., Ateş, and C., Yeroğlu, "Harris Şahini Optimizasyon Algoritması ile Aktif Güç Kayıplarının Minimizasyonu," *Dokuz Eylül Üniversitesi Mühendislik Fakültesi Fen ve Mühendislik Dergisi*, vol. 22, no. 65, pp. 481-490, 2020. DOI: 10.21205/deufmd.2020226516.

[19] A., Kaveh, and S. Talatahari, "A Novel Heuristic Optimization Method: Charged System Search," *Acta Mechanica*, vol. 213, no. 3, pp. 267-289, 2010. DOI: 10.1007/s00707-009-0270-4.

[20] F. E., Durak, B., Hiçdurmaz, and S., Özyön, "The Design of Bessel Type High-pass Active Filter with Charged System Search Algorithm," *International Scientific and Vocational Journal*, vol. 3, no. 2, pp. 76-84, 2019.