INTERNATIONAL SCIENTIFIC AND VOCATIONAL JOURNAL (ISVOS JOURNAL)

Vol.: 9 Issue: 1 Date: 30.06.2025 Received: 17.03.2025 Accepted: 30.04.2025 Final Version: 27.06.2025

ISVOS Journal, 2025, 9(1): 42-67 – DOI: 10.47897/bilmes.1659488

Spider Wasp 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 Spider Wasp Optimization (SWO) algorithm, 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 with minimal cost within certain constraints. Numerous optimization algorithms have been designed in the literature and used to obtain the best solutions for specific problems. The most critical aspects in solving these problems include correctly modeling the problem, determining the problem's parameters and constraints, and finally selecting an appropriate meta-heuristic algorithm to solve the objective function. Not every algorithm is suitable for every problem structure. Some algorithms perform better on fixed-dimension test functions, while others in solving variable-dimension test functions. In this study, the performance of the SWO algorithm was evaluated on 10 test functions previously used in the literature, consisting of three fixed-dimension functions (Schaffer, Himmelblau and Kowalik Functions) and seven variable-dimension functions, including one unimodal function (Elliptic Function) and six multimodal functions (Non-Continuous Rastrigin, Alpine, Levy, Weierstrass, Michalewicz, and Dixon & Price Functions). 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 (CSS), and the Backtracking Search Optimization Algorithm (BSA).

Keywords: "Spider wasp optimization, Harris hawks optimizer, charged system search algorithm, backtracking search optimization, fixed and variable size unimodal and multimodal test functions."

1. Giriş

Bir problemin optimizasyonu, verilen kısıtlar altında o problem için mevcut olan çözümler arasında en iyi olana ulaşmak demektir, kısaca en iyi çözümü bulmaktır. Optimizasyon problemleri mühendislik, tıp, karar verme ve tarım gibi birçok alanda yaygın hale gelmiştir[1-17]. Bu alanlardaki optimizasyon problemlerini çözmek için deterministik yöntemler olarak da bilinen çeşitli geleneksel yöntemler önerilmiştir. Ancak bu yöntemlerin yerel minimumlara yakalanma, gradyan (türev) bilgisi gerektirme, uzun zaman alma ve bazı gerçek dünya optimizasyon problemlerine uygulanamaması gibi dezavantajları vardır. Bu nedenle son kırk yıldan beridir bu problemlerin üstesinden gelmek için çeşitli meta-sezgisel algoritmalar (MA) olarak bilinen modern stokastik teknikler önerilmiştir. Meta-sezgisel algoritmalar tipik olarak biyolojik veya fiziksel olayları taklit eder, doğadan ilham alan MA'lar olarak da bilinir. MA'lar fizik tabanlı, evrim tabanlı, sürü tabanlı ve insan tabanlı olmak üzere dört ana kategoriye ayrılabilir. İlk kategori esas olarak fizik tabanlı MA'lar olarak bilinen MA'ları önermek için fizik kurallarının benzetimi kullanır. Evrim tabanlı MA'lar olarak adlandırılan ikinci kategori, doğadaki doğal evrim sürecinin benzetimini işler. Sürü tabanlı MA'lar olarak bilinen üçüncü kategori, hayvanlarda ve kuşlarda sürü zekasının sosyal davranışlarının benzetimini kullanır. Son kategori ise insanların yaşamlarındaki sosyal davranışlarından esinlenen insan tabanlı MA'lardır [18].

Çalışmada, stokastik optimizasyon için doğadan ilham alan yeni bir sürü tabanlı meta-sezgisel algoritma, yani dişi örümcek arısının avlanma, yuvalama ve çiftleşme davranışlarını taklit etmeye dayanan örümcek arısı optimizasyonu (SWO-Spider Wasp Optimizer) kullanılmıştır. Bu algoritma, örümceği arama, düşen örümceği takip etme ve kaçma, felçli örümceği yuvalama ve dişi örümcek arılarının yumurtasını bırakmak için çiftleşme davranışının benzetimini kullanan dört operatörü içermektedir.

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

Bu çalışmada SWO algoritması toplamda on adet sabit ve değişken boyutlara sahip unimodal ve multimodal test fonksiyonuna uygulanmıştır. SWO'nun uygulanmasıyla elde edilen sonuçları karşılaştırmak için literatürden üç farklı algoritma seçilerek aynı fonksiyonlara aynı şartlarda uygulanmıştır. Bu çalışma için söz konusu algoritmalar, sürü tabanlı olan Harris şahini optimize edici (HHO-Harris Hawks Optimizer) [19], fizik tabanlı olan yüklü sistem arama algoritması (CSS-Charged System Search Algorithm) [20] ve evrim tabanlı olan geri izleme arama optimizasyon (BSA-Backtracking Search Optimization Algorithm) [21] algoritması olarak belirlemiştir.

Bu çalışmanın literatüre üç temel katkısı bulunmaktadır:

(i) Çalışmada, SWO algoritmasının sabit ve değişken boyutlu, unimodal ve multimodal 10 farklı test fonksiyonu üzerindeki performansı sistematik olarak analiz edilmiştir.

(ii) SWO'nun performansı, farklı temellere dayanan (sürü, fizik ve evrim tabanlı) üç güçlü algoritma ile (HHO, CSS, BSA) kapsamlı biçimde karşılaştırılmıştır.

(iii) Algoritma ile daha önce kullanılmayan farklı test fonksiyonları optimize edilerek, algoritmanın en iyi sonuç verdiği matematiksel model tipi tartışılmıştır. Bu yönleriyle çalışma, SWO'nun farklı test fonksiyonları türlerinde kullanılabilirliğini ortaya koymuştur.

1.1. İlgili Çalışmalar

Sui ve arkadaşları çalışmalarında, Çok Stratejili Geliştirilmiş Örümcek Yaban Arısı Optimizasyonu (MISWO) adlı yeni bir algoritmayı önererek mühendislik optimizasyon problemlerine çözüm sunmaktadır. Gri Kurt Algoritması (GWO), adaptif adım boyutu ve Gauss mutasyonu gibi tekniklerle klasik Örümcek Yaban Arısı Optimizasyonu (SWO) algoritmasının performansı iyileştirilmiştir. 23 test fonksiyonu ve 7 mühendislik optimizasyon problemi üzerinde yapılan deneyler, daha hızlı yakınsama, yüksek doğruluk ve stabil performans açısından MISWO'nun diğer optimizasyon algoritmalarına kıyasla üstün sonuçlar verdiğini göstermektedir [22].

Çetinbaş çalışmasında, fotovoltaik (PV) sistemlerin modelleme doğruluğunu artırmak amacıyla Ağırlıklı Lider Arama Algoritmasını (WLS) kullanarak tek, çift ve üç diyotlu PV modellerinin parametre çıkarımını optimize etmiştir. Çalışmada, iki farklı PV hücresine ait 6 model ve altı PV modülüne ait 18 model olmak üzere toplam 24 model analiz edilmiş, WLS algoritmasının etkinliği Örümcek Yaban Arısı Optimizasyonu, Karides ve Gobi Arama, Fennec Tilkisi Optimizasyonu, Kepler Optimizasyonu gibi yedi yeni optimizasyon algoritması ile karşılaştırılmıştır. Friedman ve Wilcoxon sıralı-rank testleri ile yapılan analizler, WLS algoritmasının hata oranı, RMSE ve standart sapma açısından üstün performans gösterdiğini ortaya koymuştur [23].

Çalışmada Liang ve arkadaşları, insansız hava araçları (UAV) için üç boyutlu yol planlamasında Geliştirilmiş Örümcek Yaban Arısı Optimizasyonu (ISWO) algoritmasını önermektedir. SWO algoritmasının nüfus güncelleme formülünü iyileştirerek, Diferansiyel Evrim (DE) ve Kerevit Optimizasyonu Algoritması (COA) ile birleştirmiştir. Ayrıca, karşıt öğrenme stratejisi ile nüfus çeşitliliğini artırarak yerel minimumlara takılma problemini azaltmıştır. 2017 test seti kullanılarak, algoritma dağlık arazilerde UAV rotalarını en aza indirgeme amacıyla test edilmiştir. Sonuçlar, ISWO'nun daha az iterasyonla daha kısa ve daha güvenli uçuş yolları ürettiğini göstermektedir [24].

Bütün bu çalışmalara ek olarak, SWO algoritması ve bu algoritmanın çeşitli metotlarla geliştirilmiş versiyonları son dönemde literatürde farklı alanlarda birçok problemin çözümünde başarıyla uygulanmıştır [25-30].

2. Doğadaki Örümcek Arısı

Dünyada çoğunlukla tropikal bölgelerde yaygın olarak bulunan örümcek yaban arıları (Pompilidae), Hymenoptera'daki en büyük sivri uçlu yaban arılarından biridir. Dünyada 254 cins ve beş alt familyaya ayrılmış yaklaşık 5000 tanımlanmış Pompilidae familyasında tür vardır. Örümcek arısı, örümceği yavruları için en uygun besin kaynağı olarak kullanan yırtıcı böceklerdir. İsimleri tercih ettikleri avlardan ve benzersiz avlanma davranışlarından gelmektedir. Bu örümcek arısı türü, ince gövdeleri ve dikenli bacaklarıyla karakterize edilir. Bu örümcek arısı türünün birkaç canlı renkli türü vardır, ancak bunlar genellikle mavi veya siyahtır, bazen de metalik parlak olabilir. Siyah renkli dişi bir yaban arısı türü Şekil 1a'da verilmiştir. Bu yaban arıları genellikle çiçek nektarıyla beslenirler veya yerde av ararlar. Dişi örümcek arıları larvaları için yeterli yiyecek sağlamak amacıyla örümcekleri avlarlar [18, 22, 24-30].

Şekil 1. (a) Bir Örümcek arısı dişisi; (b) Dişi örümcek arısı felçli bir örümceği açık yuvasına doğru sürükler; (c) Erkek ve dişi örümcek yaban arıları. Dişinin önemli ölçüde daha büyük vücut büyüklüğüne dikkat edin [18].

Dişi örümcek arısı, beslenmek için, kendilerine yeterli avını (örümcek) aramak ve yumurtalarını bırakmak için, yuva aramak için kullanılan benzersiz avlanma ve yuvalama davranışlarına sahiptir. Avlanma davranışı açısından dişi örümcek arısı, çevredeki ortamlarda avlarını rastgele aramak için çeşitli besin arama stratejileri kullanır. Dişi örümcek arısı, larvalarına ev sahipliği yapacak (konakçı) örümcekleri rastgele aramaya başlar. Kısa yolculukların yanı sıra yerde veya yapraklar üzerinde titreyen kanatları ve antenleriyle sürekli hareket halinde yürürler. Uygun bir konakçı örümcek bulunduğunda, dişi örümcek arısı koşarak veya uçarak onu ağının merkezinde kovalamaya başlar ve göğsünün (sefalotoraksının) alt kısmından bir veya daha fazla zehirli ısırıkla onu bastırır. Bu durumda örümcek, ağından yere düşebilir ve larvalarını beslemek için onu felç etmek üzere dişi örümcek arısı tarafından takip edilebilir. Dişi örümcek iğnesini, konakçı olacak örümceğin hareket kabiliyetini etkileyen önemli bir sinir merkezinin yakınına yerleştirir. Bu durumda av ölmez, fakat kalıcı felç halinde kalır. Felç olan örümceğin hareketlerinin durmasından sonra örümcek arısı antenleriyle bu örümceğin yüzeyini keşfetmeye başlar. Daha sonra örümcek arısı felçli örümceği önceden hazırlanmış yuvaya çeker ve altçenesiyle bacakları arasında tutar. Avını (örümceği) önceden hazırlanmış bir yuva üzerinde sürükleyen bir örümcek yaban arısını Şekil 1 (b)'de gösterilmektedir. Ön ayaklarıyla toprağı geriye doğru kazıyarak yuva yapar. Dişi bir örümcek arısı, örümceğin karnına bir yumurta bırakır ve ardından yuvayı kapatır. Birkaç gün sonra örümcek arısının yumurtası çatlar ve konağın (örümceğin) yenilebilir kısımlarını alt çenesini kullanarak tüketmeye başlar. Örümcek arısı larvası, felçli örümceğin üzerinde birkaç gün boyunca gelişir, yetişkin oluncaya kadar çeşitli gelişim aşamalarından geçer ve birkaç hafta sonra yuvadan çıkar. Konakçı örümceğin boyutu, gelişen örümcek arısı larvalarına yiyecek sağlamaya yeterli olmalıdır. Örümcek arıları toprağı kazmak ve içinde hücre inşa etmek veya yaprak veya kayalarda çamurdan yuvalar yapmak gibi çeşitli yuvalama davranışlarına sahiptir. Diğerleri ise örümcek (av) yuvaları veya böcek delikleri gibi önceden var olan yuvaları veya oyukları kullanırlar. Dişi örümcek arısı, av örümceğinin boyutuna bağlı olarak yavrularının cinsiyetini belirleyebilir. Dişi bir örümcek arısı, haplodiploid cinsiyet belirleme sistemini (tüm zar kanatlılarda bulunur) kullanarak yumurtlama sırasında yavrularının cinsiyetini seçebilir. Bu kavram, eğer büyük konaklarda (örümcekler) büyümek dişilerde erkeklerden daha fazla üreme başarısı sağlıyorsa, annelerin çoğunlukla küçük konaklara döllenmemiş yumurtalar (erkekler) ve büyük konaklara döllenmiş yumurtalar (dişiler) bırakması gerektiği anlamına gelir. Bu nedenle, erkeklerin boyutu dişilere kıyasla daha küçüktür. Erkek ve dişi örümcek arısı boyutu Şekil 1.c'de gösterilmektedir. Konağın(örümceğin) ağırlığı larvaların büyümesi ve hayatta kalması için gereklidir. Örümcek arısının hayatta kalamaması, larvaları besleyecek yeterli büyüklükte bir örümceğin bulunmamasından kaynaklanmaktadır. Küçük bir örümcek konakçı, örümcek arısı larvasına yeterli miktarda yiyecek sağlamayacaktır. Örümcek arılarının kuru bir alanda, özellikle kayaların üzerinde yuva yapması, başka bir böceğin yuvayı keşfetmesi ve larvayı ev sahibiyle birlikte yerken onu yok etmeye çalışması, ayrıca bilinmeyen başka nedenler de dahil olmak üzere çeşitli dış nedenler yuva kurma ve larva büyütme başarısızlığına yol açabilir [18, 22, 24-30].

3. Yöntem: Örümcek Arısı Optimizasyon Algoritması

Bu çalışmada, her örümceğin karnına tek bir yumurta bırakarak bazı örümcek arısı türleri için avlanma ve yuva yapma davranışlarından ve zorunlu kuluçka parazitliğinden esinlenmiş, yeni bir optimizasyon algoritması olan SWO tanıtılmaktadır. Dişi örümcek arıları ilk olarak uygun örümcekler için çevredeki ortamları araştırır. Uygun avı ve yuvaları bulduktan sonra onları felç eder ve yuvalara sürükler. Dişi örümcek arıları uygun avı yuvalara sürükledikten sonra, örümceğin karnına bir yumurta bırakır ve yuvayı kapatırlar. SWO algoritması, arama uzayına belirli sayıda dişi örümcek arısını rastgele dağıtır. Sonra dişi örümcek arılarının her biri, tüm zar kanatlılarda bulunan avlanma ve takip etme davranışları olarak bilinen haplodiploid cinsiyet belirleme sistemi tarafından belirlenen yavrularının cinsiyetine uygun bir örümcek aramak için sürekli hareket halinde arama alanını keşfetmeye devam edeceklerdir. Dişi örümcek arıları, uygun örümcekleri bulduktan sonra ağın merkezlerinde onları yiyecekler ve ağdan düşen örümcekleri ise altı kez yerde arayacaktır. Daha sonra dişi örümcek arısı avına saldırır ve onu önceden hazırlanmış yuvaya sürüklemek için felç etmeye çalışır. Daha sonra dişi örümcek arısı örümceğin karnına bir yumurta bırakarak yuvayı kapatır. Bu algoritma kapsamında arama, takip etme ve kaçma, yuvalama ve çiftleşme davranışları olarak belirtilen örümcek arısı davranışlarının benzetimi gerçekleştirilerek matematiksel modelleri oluşturulacaktır [18, 22, 24-31].

3.1. Başlangıç Popülasyonunun Oluşturulması

Bu algoritmada, her bir örümcek arısı (dişi), mevcut nesildeki bir çözümü temsil eder ve *D* boyutlu vektör olarak eşitlik (1)'deki gibi kodlanır.

$$\overrightarrow{SW} = [x_1, x_2, x_3, \ldots, x_D] \tag{1}$$

Üst başlangıç parametre sınırı olan $\vec{H}$ ile alt başlangıç parametre sınırı olan $\vec{L}$ değerleri önceden belirlenir ve bu iki değer arasında bir dizi *N* vektörü eşitlik (2)'deki gibi rastgele oluşturulur [18, 22, 24-30]:

$$SW_{Pop} = \begin{bmatrix} sw_{1,1} & sw_{1,2} & \cdots & sw_{1,D} \\ sw_{2,1} & sw_{2,2} & \cdots & sw_{2,D} \\ \vdots & \vdots & \vdots & \vdots \\ sw_{N,1} & sw_{N,2} & \cdots & sw_{N,D} \end{bmatrix} \tag{2}$$

Burada SW_{Pop} örümcek arılarının başlangıç popülasyonudur. Arama alanında rastgele herhangi bir çözüm oluşturmak için aşağıdaki denklem kullanılır.

$$\overrightarrow{SW}_i^t = \vec{L} + \vec{r} \times \left(\vec{H} - \vec{L}\right) \tag{3}$$

Burada *t* nesil, *i* nüfus indeksini göstermektedir ($i=1,2,\ldots,N$). Denklemdeki $\vec{r}$ise 0 ile 1 arasında rastgele başlatılan D boyutlu sayılardan oluşan bir vektörü temsil etmektedir.

3.2. Avlanma ve Yuvalama Davranışı

Dişi örümcek arısı, av yolculuğuna larvalarını besleyecek örümceği/avı arayarak başlar. Aramaları, en uygun avı bulmak için arama alanı içerisinde rastgele gerçekleştirilir ve buna arama veya keşif aşaması denir. Sonra avının etrafını sarar ve koşarak ya da uçarak onu kovalar. Bu aşamaya kuşatma ve kovalama aşaması denir. Son olarak, son aşamada ise örümcek arısı yumurtasını felçli örümceğin karnının üzerine bırakmak için onu önceden hazırlanmış yuvaya doğru sürükler [18, 22, 24-30].

3.2.1. Arama Aşaması (Keşif)

Bu aşama, dişi örümcek arılarının, larvalarını beslemek için en uygun örümcekleri bulma davranışlarının benzetimini içerir. Bu aşamada dişi örümcek arısı, yavrularına uygun örümceği bulmak için arama alanını sabit bir adımla rastgele keşfeder. Bu davranış, her dişi örümcek arısının mevcut konumunu her *t* neslinde sabit bir hareketle güncelleyen ve dişi örümcek arısının keşif davranışının benzetimini içeren Denklem (4)'teki gibi modellenir [18, 22, 24-30].

$$\overrightarrow{SW}_i^{t+1} = \overrightarrow{SW}_i^t + \mu_1 \times \left(\overrightarrow{SW}_a^t - \overrightarrow{SW}_b^t\right) \tag{4}$$

Burada *a* ve *b* keşif yönünü belirlemek için popülasyondan rastgele seçilen iki indekstir ve bunları dişi örümcek arısı takip eder ve Denklem (5)'teki μ_1 ise o anki mevcut yöndeki sabit hareketi belirlemek için kullanılır. Denklemdeki r_1 sıfır ile bir aralığında ve *rn* ise normal dağılım kullanılarak oluşturulmuş rastgele sayıları göstermektedir.

$$\mu_1 = |rn| \times r_1 \tag{5}$$

Dişi örümcek arıları bazen ağdan düşen örümceğin izini kaybederler ve bu nedenle örümceğin düştüğü noktanın çevrelediği tüm bölgeyi ararlar. Bu davranışa göre Denklem (4)'teki eşitlikten farklı olarak Denklem (6)'da ifade edilen yeni ve farklı keşif yöntemi oluşturulmuştur. Bunun amacı, algoritmanın düşen örümceğin etrafındaki bölgeleri küçük bir adım büyüklüğü ile keşfetmesini sağlamaktır. Bu denklem ayrıca, düşen örümceğin konumunu temsil etmek üzere popülasyondan rastgele seçilen bir dişi örümcek arısının konumuna göre mevcut dişi örümcek arısının her iterasyonda sabit bir hareketle güncellenmesine dayanmaktadır. Denklemlerdeki *c* popülasyondan rastgele seçilen bir indeksi, *l* ise (-2) ile 1 arasında rastgele oluşturulan bir sayıyı ifade etmektedir [18, 22, 24-30].

$$\overrightarrow{SW}_i^{t+1} = \overrightarrow{SW}_c^{t+1} + \mu_2 \times \left(\vec{L} + \overrightarrow{r_2} \times \left(\vec{H} - \vec{L}\right)\right) \tag{6}$$

$$\mu_2 = B \times cos(2\pi l) \tag{7}$$

$$B = \frac{1}{1 + e^l} \tag{8}$$

Denklem (4) ve (6), arama alanını keşfetmek ve en umut verici bölgeleri bulmak için birbirlerini tamamlamaktadırlar. Ancak, dişi örümcek arısının bir sonraki pozisyonunu oluşturmak için bir denklem kullanılması gerektiği düşünüldüğünde, acaba bu durumda Denklem (4) mü, yoksa (6) mı kullanılmalıdır? Bu sorunun cevabı ise Denklem (9) kullanılarak rastgele belirlenmiş olur ve denklemdeki r_3 ve r_4 [0, 1] aralığında iki rastgele sayı ifade etmektedir.

$$\overrightarrow{SW_i}^{t+1} = \begin{Bmatrix} Denklem\ (4),\ eğer\ r_3 < r_4 \\ Denklem\ (6),\ diğer\ durumlarda \end{Bmatrix} \tag{9}$$

3.2.2. Takip ve Kaçış Aşaması (Keşif ve Kullanım)

Örümcek arısı avını bulduktan sonra ağın göbeğinde onlara saldırmaya çalışır, ancak örümcekler kaçmak için yere düşerler. Örümcek arısı düşen örümcekleri takip ederek onları felç eder ve önceden hazırlanmış yuvalarına sürükler. Diğer durumlarda örümcek, ağın merkezinden düşen örümceklerin izini kaybeder. Örümcek arısının bu davranışı bir yandan kaçarken bir yandan da örümceği yakalamaya çalıştığı anlamına gelir. Bu davranış biçimi iki eğilimin benzetimini içerir. Birincisi, örümcek arısı örümcekleri tuzağa düşürmek için avlamasıdır. Bu durumda, örümcek arısının avını takip edecek şekilde Denklem (10) uygulanarak konumu güncellenir. İkinci davranış ise mevcut iterasyon sayısı arttıkça aralarındaki mesafeyi artırmak için bir mesafe faktörü tasarlayarak örümcek arılarından kaçmanın benzetimini içerir. Bu nedenle av, örümcek arısından uzak bazı bölgelerde saklanabilir. Matematiksel olarak ifade edilen Denklem (10)'daki ilk eğilim, örümcek arısının örümcekleri yakalamak için yaptığı kovalama işleminin benzetimini içerir. Burada av ile örümcek arıları arasındaki mesafe başlangıçta küçüktür ve örümcek arıları ile avın hızına göre artabilir veya azalabilir. Denklem (10)'daki C, 2 değeri ile başlayıp doğrusal olarak sıfıra düşen örümcek arısının hızını belirlemek için kullanılan bir mesafe kontrol faktörüdür. Eğer $C > 0.5$ ise örümcek arısı avdan daha hızlıdır, ya da $C < 0.5$ ise av örümcek arısından daha hızlıdır demektir. Aşağıdaki denklemlerde a popülasyondan rastgele seçilen bir indeksi, t ve t_{max} sırasıyla mevcut ve maksimum değerlendirmeyi belirtmektedir. $\overrightarrow{r_5}$ ise [0, 1] aralığında rastgele üretilen değerleri temsil eden bir vektörü ve r_6 ise [0, 1] aralığında rastgele üretilen bir sayıyı göstermektedir [18, 22, 24-30].

$$\overrightarrow{SW_i}^{t+1} = \overrightarrow{SW_i}^{t} + C \times \left|2 \times \overrightarrow{r_5} \times \overrightarrow{SW_a}^{t} - \overrightarrow{SW_i}^{t}\right| \tag{10}$$

$$C = \left(2 - 2 \times \left(\frac{t}{t_{max}}\right) \times r_6\right) \tag{11}$$

Bir örümcek, dişi örümcek arısından kaçtığında aralarındaki mesafe giderek artar. Bu aşama başlangıçta kullanımdır (exploitation). Mesafenin artmasıyla birlikte kullanım keşfe dönüşür. Bu davranışın benzetimi Denklem (12)'deki formül kullanılarak yapılır. Denklemde $\overrightarrow{vc}$ normal dağılıma göre (k) ile (-k) değerleri arasında oluşturulan bir vektördür. Dişi örümcek arısı ile örümcek arasındaki mesafeyi kademeli olarak artırmak için k, Denklem (13) kullanılarak üretilir. Bu iki eğilim arasındaki denge, Denklem (14)'te gösterildiği gibi rastgele elde edilir.

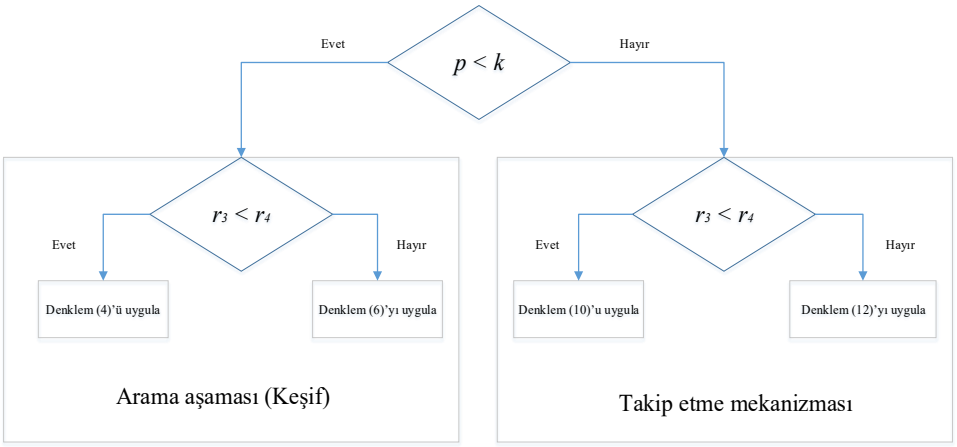
$$\overrightarrow{SW_i}^{t+1} = \overrightarrow{SW_i}^{t} \times \overrightarrow{vc} \tag{12}$$

$$k = 1 - \left(\frac{t}{t_{max}}\right) \tag{13}$$

$$\overrightarrow{SW_i}^{t+1} = \begin{Bmatrix} Denklem\ (10), & eğer \quad r_3 < r_4 \\ Denklem\ (12), & diğer\ durumlarda \end{Bmatrix} \tag{14}$$

Optimizasyon sürecinin başlangıcında, tüm örümcek arıları, optimuma yakın çözümü içerebilecek umut verici bölgeyi bulmak için optimizasyon problemlerinin etki alanını küresel olarak aramak için keşif mekanizmasını uygulayacaktır. Algoritma, yerel minimumda takılıp kalmaktan kaçınmak için iterasyon geçişi sırasında mevcut örümcek arılarının etrafındaki alanları keşfetmek ve kullanmak için takip etme ve kaçma mekanizmasını kullanacaktır. Son olarak, arama aşaması ile takip mekanizması arasındaki değişim aşağıdaki formüle göre ayarlanır. Burada p, [0, 1] arasında rastgele bir sayıdır. Arama aşaması ile takip etme ve kaçış mekanizması arasındaki ilişki, Şekil 2'de gösterilmiştir [18, 22, 24-30]:

$$\overrightarrow{SW_i}^{t+1} = \begin{Bmatrix} Denklem\ (9), & eğer \quad p < k \\ Denklem\ (14), & \text{diğer durumlarda} \end{Bmatrix} \tag{15}$$



Şekil 2. Arama aşaması ile takip etme ve kaçış mekanizması arasındaki denge [18].

3.2.3. Yuvalama Davranışı (Kullanım)

Dişi örümcek arısı felçli örümceği önceden hazırlanmış olan yuvaya çeker. Örümcek arısı, toprağı kazmak ve toprakta hücreler oluşturmak, yapraklarda veya kayalarda çamurdan yuvalar yapmak veya örümcek (av) yuvaları veya böcek delikleri gibi önceden var olan yuvaları veya oyukları kullanmak gibi çok sayıda yuvalama davranışına sahiptir. Örümcek arısı çeşitli yuvalanma davranışlarına sahip olduğundan, bu algoritmada söz konusu davranışların iki farklı denklemle benzetimi gerçekleştirilmiştir. Denklem (16), örümceğin en uygun örümceğin bulunduğu bölgeye doğru çekilmesi ve felçli örümceğin yerleştirilip karnının üzerine yumurta bırakılması için yuva yapılabilecek en iyi yer olarak kabul edilmesi esasına dayanır. Bu ilk denklem şu şekilde açıklanmaktadır. Denklemdeki $\overrightarrow{SW}^*$ o ana kadar ki en iyi çözümü temsil eder [18, 22, 24-30].

$$\overrightarrow{SW}_i^{t+1} = \overrightarrow{SW}^* + cos(2\pi l) \times \left(\overrightarrow{SW}^* - \overrightarrow{SW}_i^{\,t}\right) \tag{16}$$

Denklem (17) ise, aynı konumda iki yuva yapmaktan kaçınmak için ek bir adım boyutu kullanarak popülasyondan rastgele seçilen bir dişi örümceğin konumu dahilinde yuvayı yapması esasına dayanır. Burada; r_3, [0, 1] aralığında oluşturulan rastgele bir sayıyı, γ levy uçuşuna göre üretilen bir sayıyı göstermektedir. Denklemdeki a, b ve c popülasyondan rastgele seçilen üç çözümün indislerini ve Denklem (18)'e göre değeri belirlenen $\vec{U}$ ise aynı konumda iki yuva inşa edilmesini önlemek için bir adım boyutunun ne zaman uygulandığını belirlemek için kullanılan ikili bir vektörü ifade etmektedir. Burada $\overrightarrow{r_4}$ ve $\overrightarrow{r_5}$ [0, 1] aralığındaki rastgele değerleri temsil eden iki vektördür.

$$\overrightarrow{SW}_i^{t+1} = \overrightarrow{SW}_a^{t} + r_3 \times |\gamma| \times \left(\overrightarrow{SW}_a^{t} - \overrightarrow{SW}_i^{t}\right) + (1 - r_3) \times \vec{U} \times \left(\overrightarrow{SW}_b^{t} - \overrightarrow{SW}_c^{t}\right) \tag{17}$$

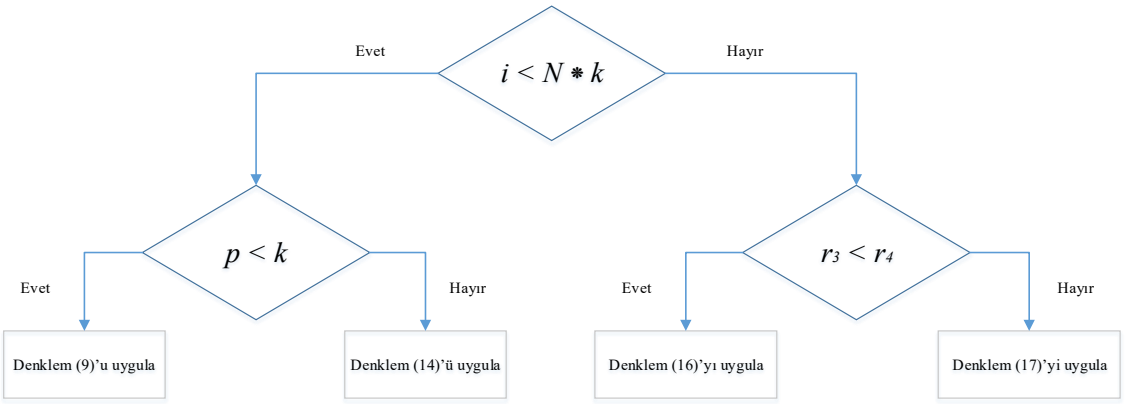
$$\vec{U} = \begin{cases} 1, & eğer \quad \overrightarrow{r_5} < \overrightarrow{r_4} \\ 0, & diğer\ durumlarda \end{cases} \tag{18}$$

Algoritmada örümcek arısı yuvalanma davranışlarının Denklem (16) ve (17) gibi iki farklı denklemle benzetimi yapıldığından hangi denklemin kullanılacağı Denklem (19)'la belirlenir.

$$\overrightarrow{SW}_i^{t+1} = \begin{cases} Denklem\ (16), & eğer \quad r_3 < r_4 \\ Denklem\ (17), & \text{diğer durumlarda} \end{cases} \tag{19}$$

Son olarak, avlanma ve yuvalama davranışları arasındaki denge, Denklem (20) kullanılarak elde edilir. Algoritmada tüm örümcek arılarının optimizasyon sürecinin başlangıcında ilgili örümceklerini araması Şekil 3'de gösterilmiştir. Sonra, örümcek arıları uygun örümcek arılarını önceden hazırlanmış yuvalara çekecektir [18].

$$\overrightarrow{SW}_i^{t+1} = \begin{cases} Denklem\ (15), & eğer \quad i < N \times k \\ Denklem\ (19), & \text{diğer durumlarda} \end{cases} \tag{20}$$



Şekil 3. SWO'da avlanma ve yuvalama davranışlarının akış şeması [18].

3.3. Çiftleşme Davranışı

Örümcek arılarının ana özelliklerinden biri cinsiyet belirleme yetenekleridir. Cinsiyet, yumurtanın bırakıldığı konağın büyüklüğüne göre belirlenir. Şekil 1.c'den görüleceği üzere erkekler küçük örümcek arısı ile temsil edilirken, dişiler büyük örümcek arısı ile temsil edilmektedir. Algoritmada, her örümcek arısı mevcut nesildeki olası bir çözümü ve örümcek arısı yumurtası da o nesildeki yeni üretilen potansiyel çözümü temsil eder. Yeni çözümler (örümcek arısı yumurtaları) aşağıdaki denkleme göre üretilir [18, 22, 24-30].

$$SW_i^{t+1} = Crossover(SW_i^t, SW_m^t, CR) \tag{21}$$

Burada; *Crossover*, SW_i^t ve SW_m^t çözümleri arasında çaprazlama oranı (*CR*- Crossover Rate) olarak bilinen bir olasılıkla uygulanan tek biçimli çaprazlama operatörünü belirtir; SW_m^t ve SW_i^t sırasıyla erkek ve dişi örümcek arısı temsil eden iki vektördür. Algoritmada erkek örümcek arısı, dişi örümcek arısından farklı olacak şekilde Denklem (22) kullanılarak oluşturulur.

$$\overrightarrow{SW_i^{t+1}} = \overrightarrow{SW_i^t} + e^l \times |\beta| \times \overrightarrow{v_1} + (1 - e^l) \times |\beta_1| \times \overrightarrow{v_2} \tag{22}$$

Burada; β ve β_1 normal dağılıma göre rastgele oluşturulmuş iki sayıyı ve *e* ise üstel sabiti göstermektedir. Denklemdeki $\overrightarrow{v_1}$ ve $\overrightarrow{v_2}$ vektörleri aşağıda verilen eşitliklere göre üretilir.

$$\overrightarrow{v_1} = \begin{cases} (\overrightarrow{x_a} - \overrightarrow{x_i}), & eğer\ f(\overrightarrow{x_a}) < f(\overrightarrow{x_i}) \\ (\overrightarrow{x_i} - \overrightarrow{x_a}), & diğer\ durumlarda \end{cases} \tag{23}$$

$$\overrightarrow{v_2} = \begin{cases} (\overrightarrow{x_b} - \overrightarrow{x_c}), & eğer\ f(\overrightarrow{x_b}) < f(\overrightarrow{x_c}) \\ (\overrightarrow{x_c} - \overrightarrow{x_b}), & diğer\ durumlarda \end{cases} \tag{24}$$

Denklemlerde *a*, *b* ve *c*, $a \neq i \neq b \neq c$ olacak şekilde popülasyondan rastgele seçilen üç çözümün indekslerini ifade etmektedir. Çaprazlama ise iki ebeveynli örümcek arısı içindeki genetik materyali yeniden birleştirerek ebeveynlerinin özelliklerini paylaşan bir yavru (yumurta) üretmek için kullanılır. Avlanma ve çiftleşme davranışları arasındaki değiş tokuş, önceden tanımlanmış bir faktör olan değiş tokuş oranına (*TR*-Tradeoff Rate) dayanmaktadır.

3.4. Popülasyon Azaltma ve Bellek Tasarrufu

Dişi örümcek, konağın karnına yumurta bıraktıktan sonra yuvayı kapatır ve yuva alanını göze çarpmayan bir şekilde bırakır. Bu durum, dişi örümceğin optimizasyon sürecindeki rolünün neredeyse tamamlandığını ve optimizasyon sürecinin geri kalanında fonksiyon değerlendirmelerini diğer örümcek arılarına devretmenin daha iyi sonuçlara ulaşmaya yardımcı olabileceği anlamına gelmektedir. İterasyon sırasında, popülasyondaki bazı örümcek arıları, diğer örümcek arılarına daha fazla fonksiyon değerlendirmesi sağlamak için sonlandırılırken, aynı zamanda popülasyon çeşitliliğini azaltarak yakınsama hızını optimum çözüme yakın bir çözüme doğru hızlandıracaktır. Bu nedenle tüm fonksiyon değerlendirmelerinin her değerlendirmesinde, yeni popülasyonun uzunluğu denklem (25) kullanılarak güncellenir [18, 22, 24-30].

$$N = N_{min} + (N - N_{min}) \times k \tag{25}$$

Burada N_{min} optimizasyon sürecinin farklı aşamalarında yerel minimumlara takılıp kalmamak için kullanılan minimum nüfus sayısını göstermektedir. Algoritmada, her bir örümcek arısı tarafından elde edilen en iyi örümcek konumunu bir sonraki nesilde güncellenmesi için bellek tasarrufu gerçekleştirilir. Kısaca her bir örümcek arısı tarafından elde edilen her bir çözüm, bir önceki

nesildeki eşdeğeri ile karşılaştırılır ve mevcut çözüm daha uygun ise yenisi ile değiştirilir. SWO algoritmasının sözde kodu Algoritma 1'de verilmiştir [18, 26].

Algoritma 1. SWO Algoritmasının sözde kodu [18, 26]

Input: N, N_{min}, CR, TR, t_{max}
Output: $\overrightarrow{SW}^*$
Denklem (3)'ü kullanarak N dişi örümcek arısını başlatma. $\overrightarrow{SW_i}(i = 1,2,\ldots,N)$
Her $\overrightarrow{SW_i}$ değerini değerlendirme ve $\overrightarrow{SW}^*$içinde en iyi uygunluk değere sahip olan değeri bulma
$t=1$; // mevcut fonksiyonu değerlendirme
while($t < t_{max}$)
  r_6: 0 ile 1 arasında rastgele değerler üretme
  if ($r_6 < TR$) %% Avlanma ve Yuvalama davranışı
    for $i=1:N$
      Şekil 4.'ü uygulama
      $f(\overrightarrow{SW_i})$ değerini hesaplama
      $t=t+1$
    End for
  Else %% Çiftleşme davranışı
    for $i=1:N$
      Denklem (21)'i uygulama
      $t=t+1$
    End for
  End if
  Memory Saving uygulama
  Denklem (25)'i kullanarak N değerlerini güncelleme
End while

4. SWO Algoritmasının Sonuçlarını Karşılaştırmak üzere Seçilen Diğer Algoritmalar

Bu bölümde SWO algoritması uygulanarak elde edilen sonuçları karşılaştırmak için kullanılan HHO, CSS ve BSA algoritmaları kısaca özetlenmiştir.

4.1. Harris Şahin Optimizasyonu (HHO) Algoritması

Bu algoritma, doğada iyi bilinen bir yırtıcı kuş olan Harris şahinlerinin avlanma stratejisi benzetimi yapılarak oluşturulmuştur [19]. Harris şahini, 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 düzenleyebilir. Yırtıcı kuşlar aleminde gerçek anlamda işbirlikçi avcılar olarak bilinirler. Harris şahinleri avlanma süreçlerinde özellikle tavşan avlama sürecinde sürü olarak hareket eder. Sürünün bir lideri (en iyi şahin) bulunur ve 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 Harris şahinleri optimizasyonu (Harris hawks optimization, HHO) algoritması olarak modellenmiştir [19]. 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 azalacağı için 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 [19].

Algoritma 2. HHO Algoritmasının sözde kodu [19]

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 ($|E|\geq 1$) **then** Keşif Aşamasındaki Konum vektörünü güncelle.
    if ($|E|<1$) **then** (Sömürü Aşaması)
      if ($r\geq 0.5$ ve $|E|\geq 0.5$) **then** Yumuşak Kuşatma Konum vektörünü güncelle.
      else if ($r\geq 0.5$ ve $|E|<0.5$) **then** Sıkı Kuşatma Konum vektörünü güncelle.
      else if ($r<0.5$ ve $|E|\geq 0.5$) **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}

4.2. 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 elleçleme yaklaşımıyla düzeltilir. Ayrıca, en iyi sonuçları kaydetmek için yüklü bellek kullanılır. CSS algoritmasının geliştirilmesi aşağıda kısaca özetlenen 8 kural üzerinden gerçekleştirilmiştir. CSS algoritmasının bu kurallar çerçevesinde sözde kodu ise Algoritma 3'te verilmiştir [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, alternatif olarak bu durumda 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 [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).

4.3. 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 [21].

Algoritma 4. BSA Algoritmasının sözde kodu [21]

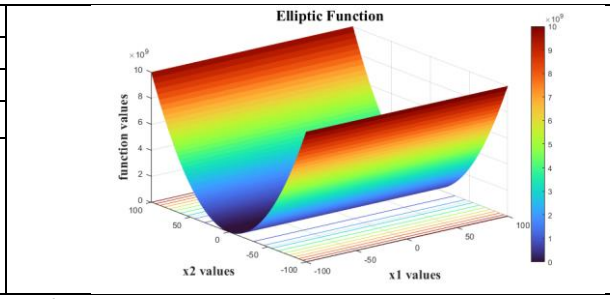
Input: *ObjFun*, *N*, *D*, *maxcycle*, *mixrate*, $low_{1:D}$, $up_{1:D}$
Output: *globalminimum*, *globalminimizer*
// *rnd* ~ *U*(0,1), *rndn* ~ *N*(0,1), *w*=*rndint*(·), *rndint*(·) ~ U(1,·) | *w* ∈ {1, 2, 3,...,·*N*}
function bsa(*ObjFun*, *N*, *D*, *maxcycle*, *low*, *up*)
// BAŞLANGIÇ
globalminimum=inf
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
        mapi,u(1:[mixrate·rnd·D]= 0 | u=permuting(1,2,3,…,D)
      end
    else
      for i from 1 to N do, mapi,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 mapi,j=1 then Ti,j:=Pi,j
      end
    end
    // Sınır Kontrol Mekanizması
    for i from 1 to N do
      for j from 1 to D do
        if (Ti,j<lowj) or (Ti,j>upj) then
          Ti,j=rnd·(upj-lowj)+lowj
        end
      end
    end
  end
  // Seçim-II
  fitnessT=ObjFun(T)
  for i from 1 to N do
    if fitnessTi<fitnessPi then
      fitnessPi:=fitnessTi
      Pi=Ti
    end
  end
  fitnessPbest=min(fitnessP) | best ∈ {1,2,3,…,N}
  if fitnessPbest<globalminimum then
    globalminimum:=fitnessPbest
    globalminimizer:=Pbest
    // Global minimum ve global minimizer dışarı aktarılır.
  end
end
```

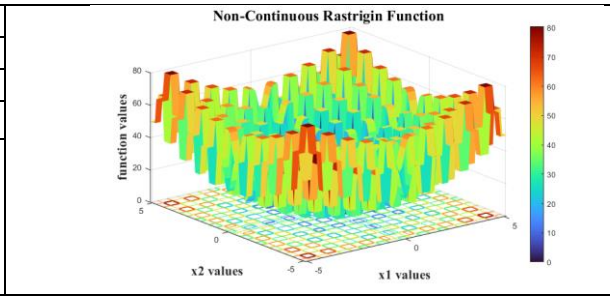
5. Test Fonksiyonları

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 Şekil 4-13 arasında gösterilmiştir [1, 31, 32].

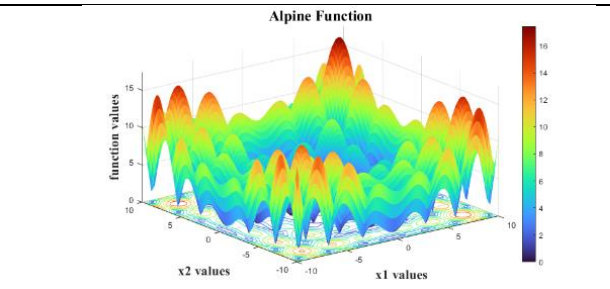
Bu çalışmada kullanılan test fonksiyonları, meta-sezgisel algoritmaların performanslarını farklı problem türlerinde değerlendirebilmek için yaygın olarak tercih edilen standart matematiksel fonksiyonlardır. f_1-f_7 arası fonksiyonlar yüksek boyutlu (30, 50 ve 100) değişken boyutlu fonksiyonlardır ve optimizasyon algoritmalarının hem küresel arama (exploration) hem de yerel arama (exploitation) kabiliyetlerini test etmek için uygundur. f_8-f_{10} arası fonksiyonlar ise sabit boyutlu fonksiyonlardır ve genellikle parametrik karmaşıklıkların az olduğu ortamlarda algoritmaların doğruluğunu ve hassasiyetini değerlendirmek için kullanılır. Her bir fonksiyonun minimum değeri, arama alanı ve boyut bilgisi tablolar ve grafiklerle birlikte aşağıda sunulmuştur. Ayrıca her fonksiyonun optimizasyon probleminde karşılık geldiği zorluk türleri de (örneğin: çoklu yerel minimum, süreksizlik, vs.) belirtilmiştir.

Fonksiyon Adı	Elliptic Function	
(*fmin*) Değeri	0	
Boyut Sayısı	30, 50, 100	
Aralık Değerleri	$(-100,100)^n$	
Yapısı	Tek modlu Ayrılamaz Sürekli Dışbükey	
$f_1(x) = \sum_{i=1}^{d} (10^6)^{\frac{i-1}{d-1}} \times x_i^2$		

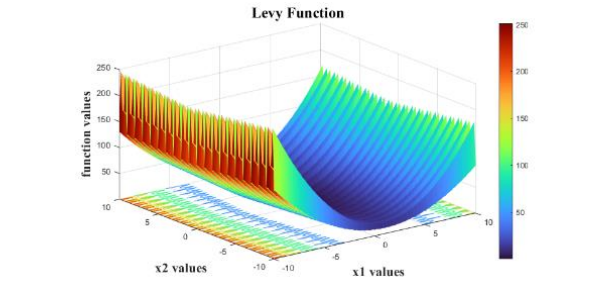
Şekil 4. ***f1*** **fonksiyonuna ait özellikler ve** ***3-D*** **arama uzayı.**

Fonksiyon Adı	Non-Continuous Rastrigin Function	
(*fmin*) Değeri	0	
Boyut Sayısı	30, 50, 100	
Aralık Değerleri	$(-5.12,5.12)^n$	
Yapısı	Sürekli Dışbükey Çok modlu Türevlenebilir Ayrılabilir	
$f_2(x) = \sum_{i=1}^{d} [x_i^2 - 10\cos(2\pi x_i) + 10], \quad x_i = \begin{cases} \text{eğer } \lvert x_i \rvert \le 0.5 & ise \quad x_i \\ \text{eğer } \lvert x_i \rvert > 0.5 & ise \quad round(2 \cdot x_i)/2 \end{cases}$		

Şekil 5. ***f2*** **fonksiyonuna ait özellikler ve** ***3-D*** **arama uzayı.**

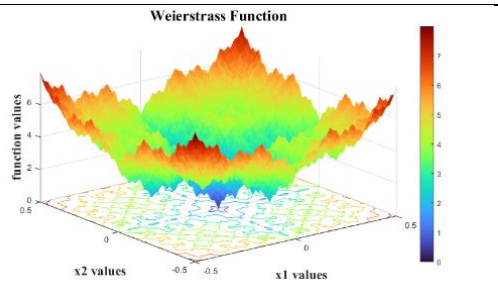
Fonksiyon Adı	Alpine Function	
(*fmin*) Değeri	0	
Boyut Sayısı	30, 50, 100	
Aralık Değerleri	$(-10,10)^n$	
Yapısı	Dışbükey değildir Ayrılamaz Türevlenebilir	
$f_3(x) = \sum_{i=1}^{d} \lvert x_i \sin(x_i) + 0.1x_i \rvert$		

Şekil 6. ***f3*** **fonksiyonuna ait özellikler ve** ***3-D*** **arama uzayı.**

Fonksiyon Adı	Levy Function	
(*fmin*) Değeri	0	
Boyut Sayısı	30, 50, 100	
Aralık Değerleri	$(-10,10)^n$	
Yapısı	Süreklidir Dışbükey değildir Çok modludur Ayrılamaz Türevlenebilir	
$f_4(x) = \sin^2(\pi w_1) + \sum_{i=1}^{d-1} (w_i - 1)^2[1 + 10\sin^2(\pi w_i + 1)] + (w_d - 1)^2[1 + \sin^2(2\pi w_d)], \quad w_i = 1 + \frac{x_i - 1}{4}$		

Şekil 7. ***f4*** **fonksiyonuna ait özellikler ve** ***3-D*** **arama uzayı.**

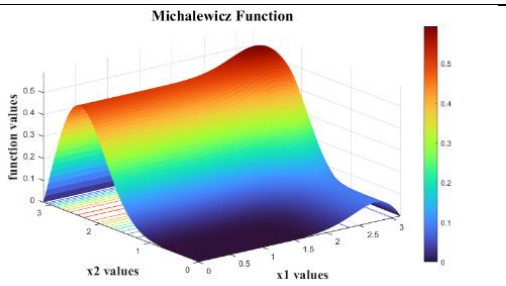
Fonksiyon Adı	Weierstrass Function
($fmin$) Değeri	0
Boyut Sayısı	30, 50, 100
Aralık Değerleri	$(-0.5,0.5)^n$
Yapısı	Süreklidir Çok modludur Ayrılamaz Türevlenebilir



$$f_5(x) = \sum_{i=1}^{d} \left[\sum_{k=0}^{k_{max}} a^k \cos\left(2\pi b^k (x_i + 0.5)\right) \right] - d \left[\sum_{k=0}^{k_{max}} a^k \cos(2\pi b^k) \right], a = 0.5, b = 3, k_{max} = 20$$

Şekil 8. ***f5*** **fonksiyonuna ait özellikler ve** ***3-D*** **arama uzayı.**

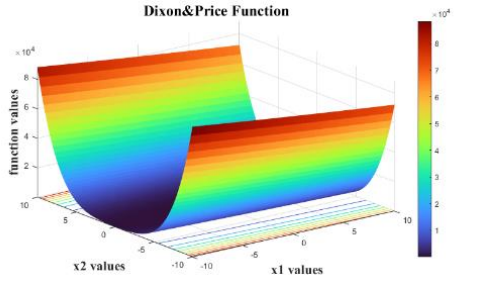
Fonksiyon Adı	Michalewicz Function
($fmin$) Değeri	0
Boyut Sayısı	30, 50, 100
Aralık Değerleri	$(0,\pi)^n$
Yapısı	Süreklidir Çok modludur Türevlenebilir Konveks değildir



$$f_6(x) = -\sum_{i=1}^{d} sin(x_i) \cdot \left(sin\left(\frac{i \cdot x_i^2}{\pi} \right) \right)^{2m}, m = 10$$

Şekil 9. ***f6*** **fonksiyonuna ait özellikler ve** ***3-D*** **arama uzayı.**

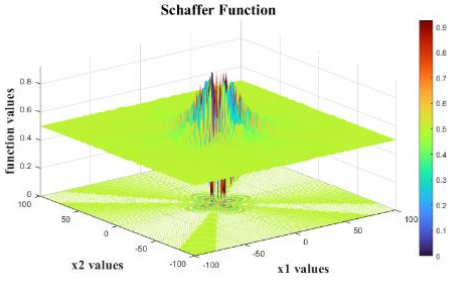
Fonksiyon Adı	Dixon&Price Function
($fmin$) Değeri	0
Boyut Sayısı	30,50,100
Aralık Değerleri	$(-10,10)^n$
Yapısı	Süreklidir Türevlenebilir Konveks değildir



$$f_7(x) = (x_1 - 1)^2 + \sum_{i=2}^{d} i \cdot (2x_i^2 - x_{i-1})^2$$

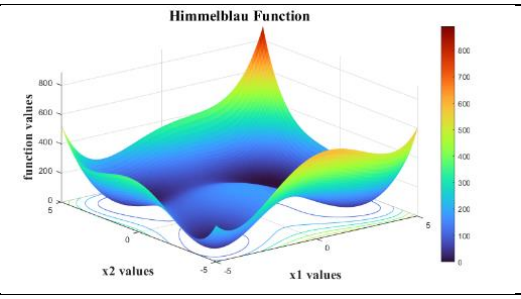
Şekil 10. ***f7*** **fonksiyonuna ait özellikler ve** ***3-D*** **arama uzayı.**

Fonksiyon Adı	Schaffer Function
($fmin$) Değeri	0
Boyut Sayısı	2
Aralık Değerleri	$(-100,100)^n$
Yapısı	Süreklidir Türevlenebilir Dışbükey değildir Tek modludur Ayrılamaz

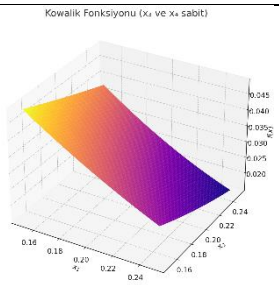


$$f_8(x, y) = 0.5 + \frac{sin^2(x^2 + y^2)^2 - 0.5}{1 + 0.001(x^2 + y^2)^2}$$

Şekil 11. ***f8*** **fonksiyonuna ait özellikler ve** ***3-D*** **arama uzayı.**

Fonksiyon Adı	Himmelblau Function	
($fmin$) Değeri	0	
Boyut Sayısı	2	
Aralık Değerleri	$(-5,5)^n$	
Yapısı	Süreklidir Türevlenebilir Dışbükey değildir	
$f_9(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$		

Şekil 12. *f9* fonksiyonuna ait özellikler ve *3-D* arama uzayı.

Fonksiyon Adı	Kowalik Function	
($fmin$) Değeri	≈ 0.0003075	
Boyut Sayısı	4	
Aralık Değerleri	$(-5,5)^n$	
Yapısı	Süreklidir Türevlenebilir Konveks değildir	
$f_{10}(x_1,x_2,x_3,x_4) = \sum_{k=1}^{11}\left[a_k - \frac{x_1 b_k(b_k + x_2)}{b_k(b_k + x_3) + x_4}\right]^2$ a=[0.1957,0.1947,0.1735,0.16,0.0844,0.627,0.456,0.342,0.323,0.235,0.246] b=[0.25,0.5,1,2,4,6,8,10,12,14,16]		

Şekil 13. *f10* fonksiyonuna ait özellikler ve *3-D* arama uzayı.

6. Sayısal Sonuçlar

Bu çalışmada yer alan test fonksiyonlarının SWO 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.

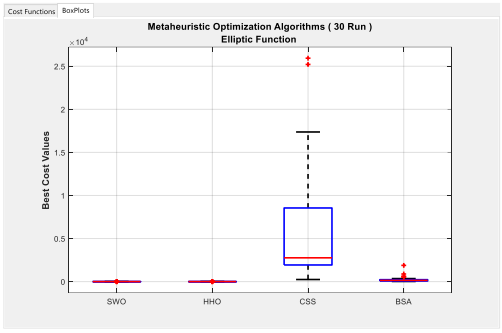
SWO algoritmasının performansını en üst düzeye çıkarmak için etkili bir şekilde tahmin edilmesi gereken minimum popülasyon büyüklüğü olan N_{min}, değiş-tokuş oranı olan TR ve çaprazlama oranı CR olan üç adet parametresi vardır. Yapılan çalışmalar sonucunda bu parametrelerin en iyi değerlerinin, TR için 0,3, CR için 0,2 ve N_{min} için ise 20 olduğu sonucuna varılmıştır [18]. 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 1'de birlikte 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 geliştirilmiştir.

Tablo 1. Çalışmada kullanılan algoritmalara ait parametreler

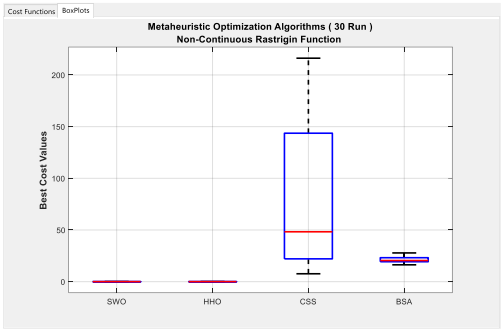
Algoritmalar	***Parametreler***			***İterasyon sayısı***	***PopN***	***FCall***
SWO	***TR***	***CR***	N_{min}	500	40	15000
	0,3	0,2	20			
HHO	***E***	J				
	[0,2]	[0,2]				
BSA	***mixrate***					
	2					
CSS	***r***	***Start k_v***	***Finish k_v***			
	0,01	0,5	0,3			
	Start k_a	***Finish k_a***				
	0,5	0,5				

6.1. Boyutu 30 Olan Test Fonksiyonları için elde edilen sonuçlar

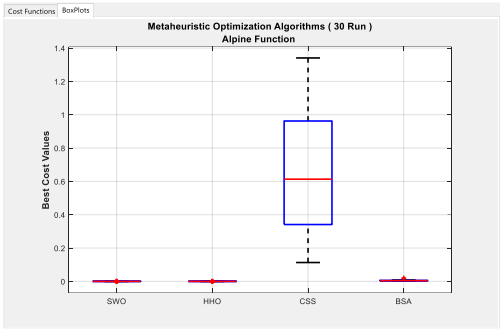
Bu bölümde 30 boyuttaki f_1-f_7 arası test fonksiyonları SWO, HHO, CSS ve BSA algoritmalarıyla 30'ar kez çözülmüş ve elde edilen kutu grafikleri Şekil 14-20 arası ve yakınsama eğrileri ise Şekil 22-28 arasında gösterilmiştir. Tüm algoritmalara göre bulunan sonuçlar en kötü, ortalama, en iyi, standart sapma ve süre olarak Tablo 2'de verilmiştir.



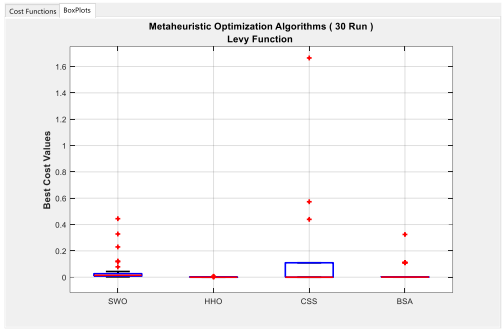
Şekil 14. f1 için kutu grafiği (30-D)



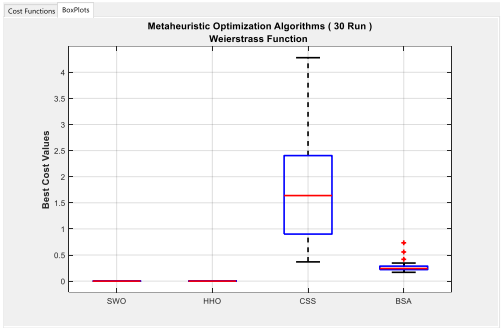
Şekil 15. f2 için kutu grafiği (30-D)



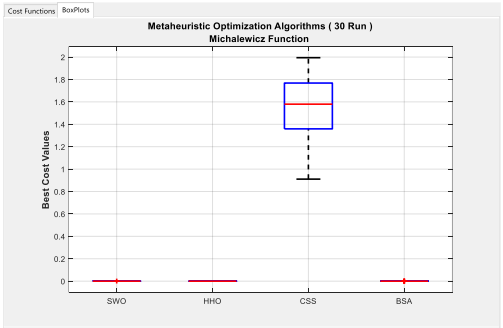
Şekil 16. f3 için kutu grafiği (30-D)



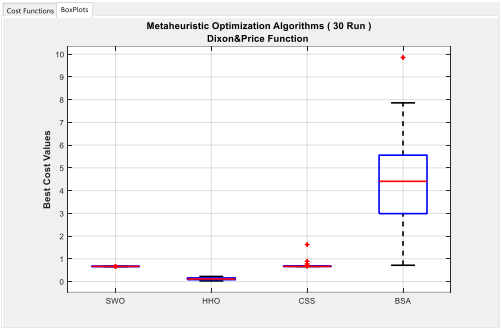
Şekil 17. f4 için kutu grafiği (30-D)



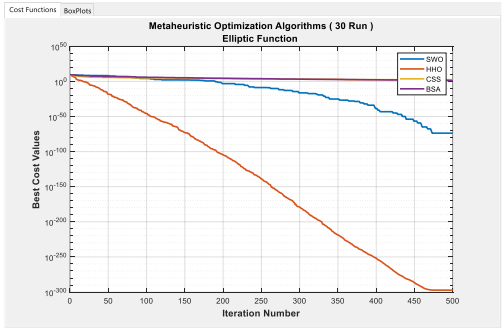
Şekil 18. f5 için kutu grafiği (30-D)



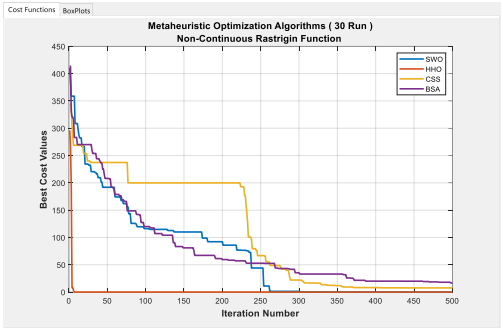
Şekil 19. f6 için kutu grafiği (30-D)



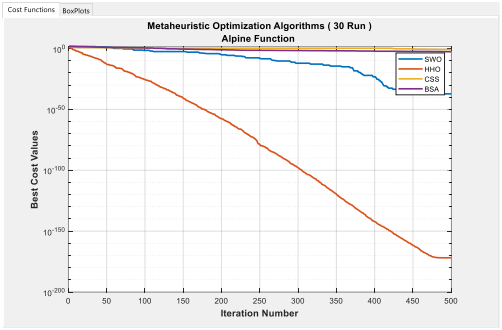
Şekil 20. f7 için kutu grafiği (30-D)



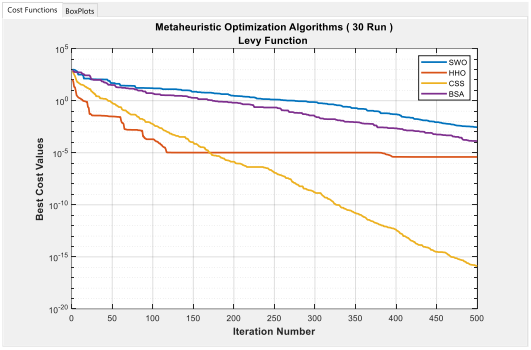
Şekil 21. f1 için yakınsama eğrileri (30-D)



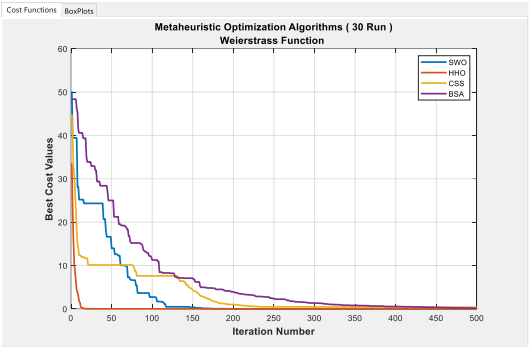
Şekil 22. f2 için yakınsama eğrileri (30-D)



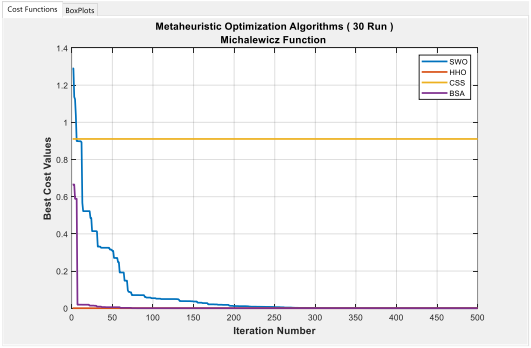
Şekil 23. f3 için yakınsama eğrileri (30-D)



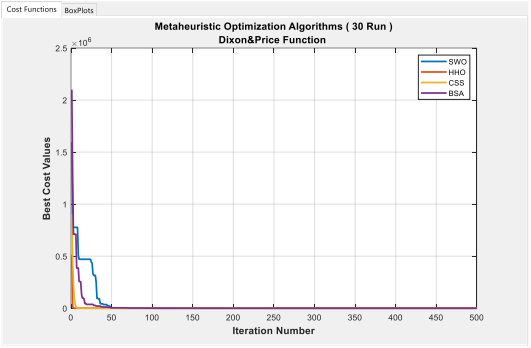
Şekil 24. f4 için yakınsama eğrileri (30-D)



Şekil 25. f5 için yakınsama eğrileri (30-D)



Şekil 26. f_6 için yakınsama eğrileri (30-D)



Şekil 27. f7 için yakınsama eğrileri (30-D)

Şekil 14-27 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 14-15'de SWO ve HHO algoritmalarının diğerlerine göre iyi bir dağılımlara sahip oldukları ve Şekil 21-22'de ise HHO'nun SWO'ya göre daha hızlı bir şekilde yakınsadığı görülmektedir. f_3 test fonksiyonu için Şekil 16'da SWO, HHO ve BSA algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 23'te ise HHO'nun SWO'ya göre daha iyi performans gösterdiği söylenebilir. f_4 test fonksiyonu için Şekil 17'de HHO algoritmasının en iyi dağılıma sahip olduğu ve Şekil 24'te ise CSS algoritmasının HHO ve SWO'ya göre daha iyi performans gösterdiği ifade edilebilir. f_5 test fonksiyonu için Şekil 18'de SWO ve HHO algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 25'de ise HHO algoritmasının SWO ile birlikte iyi yakınsadıkları görülmektedir. f_6 test fonksiyonu için Şekil 19'da SWO, HHO ve BSA algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 26'da ise HHO algoritmasının SWO'ya göre en az iterasyonla yakınsadığı söylenebilir. f_7 test fonksiyonu için Şekil 20'de SWO, HHO ve CSS algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 27'de ise HHO algoritmasının SWO'ya göre daha hızlı yakınsadığı ifade edilebilir.

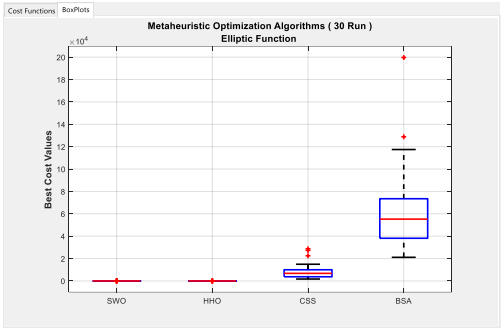
Tablo 2 incelendiğinde ise SWO algoritmasının f_2 ve f_5'de en iyi çözüm noktalarında, diğer fonksiyonlarda ise HHO ve CSS algoritmalarının en iyi değerleri yakaladıkları görülmektedir. Ortalama süre açısından da SWO algoritmasının en iyi sürede f_1, f_3, f_5, *ve* f_6'da olduğu görülmektedir. HHO algoritmasının en iyi değeri yakaladığı durumlar olan f_1, f_3, f_6 *ve* f_7'de ortalama değerler açısından karşılaştırıldığında SWO algoritmasından daha iyi olduğu ve standart sapmalar açısından karşılaştırıldığında ise f_7 hariç SWO algoritmasından daha iyi olduğu görülmektedir. CSS algoritmasının en iyi değeri yakaladığı durum olan f_4'de ortalama değerler ve standart sapmalar açısından karşılaştırıldığında ise, SWO algoritmasından daha kötü değerlere sahip olduğu görülmektedir.

Tablo 2. Algoritmaların 30 Boyutlu test fonksiyonlarına uygulanması durumunda sonuçları (30 bağımsız çalışma)

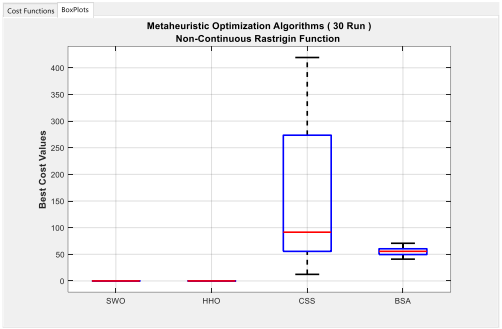
		SWO	*HHO*	*CSS*	*BSA*
f_1	EnKötü	1.397733262e-36	3.70795912e-272	2.593443940e+04	1.884212242e+03
	Ortalama	4.659110943e-38	1.23598743e-273	6.016533696e+03	2.545157701e+02
	EnİyI	2.005604575e-74	5.15654145e-298	2.441273321e+02	1.773107676e+01
	StdSapma	2.509007989e-37	0.000000000e+00	6.840519185e+03	3.622092889e+02
	Süre (s)	0.109776	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)	0.12047	0.167277	0.475873	0.105965
f_3	EnKötü	3.505053821e-22	5.14554882e-157	1.340737322e+00	1.960382296e-02
	Ortalama	1.168912126e-23	3.25080897e-158	6.553980269e-01	5.083291179e-03
	EnİyI	5.533769836e-38	1.31674151e-172	1.131546252e-01	2.038355882e-03
	StdSapma	6.291660289e-23	9.88896553e-158	3.244607945e-01	3.216886321e-03
	Süre (s)	0.0924699	0.0967534	0.373527	0.113696
f_4	EnKötü	4.443827991e-01	7.149647793e-03	1.664973966e+00	3.247722878e-01
	Ortalama	5.489774096e-02	8.214041140e-04	1.258813204e-01	3.014839083e-02
	EnİyI	2.864129340e-03	3.956924182e-06	1.211753808e-16	1.329231841e-04
	StdSapma	1.015276885e-01	1.584175480e-03	3.130086335e-01	6.827050649e-02
	Süre (s)	0.126724	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)	0.948987	3.53288	2.09585	1.81953
f_6	EnKötü	8.598985752e-04	0.000000000e+00	1.994312920e+00	2.492185246e-08
	Ortalama	1.444335406e-04	0.000000000e+00	1.548527086e+00	8.307286831e-10
	EnİyI	1.276578562e-05	0.000000000e+00	9.106032516e-01	1.434947756e-23
	StdSapma	1.579055487e-04	0.000000000e+00	2.966525027e-01	4.473609377e-09
	Süre (s)	0.130058	0.376848	0.489844	0.197483
f_7	EnKötü	6.667417809e-01	2.250845284e-01	1.627529292e+00	9.855168948e+00
	Ortalama	6.666850886e-01	1.154940209e-01	7.133118136e-01	4.405471984e+00
	EnİyI	6.666677928e-01	2.882321837e-02	6.666669529e-01	7.178599397e-01
	StdSapma	1.831909903e-05	4.662332810e-02	1.752551541e-01	2.001336416e+00
	Süre (s)	0.0621574	0.135372	0.345279	0.0564814

6.2. Boyutu 50 Olan Test Fonksiyonları İçin

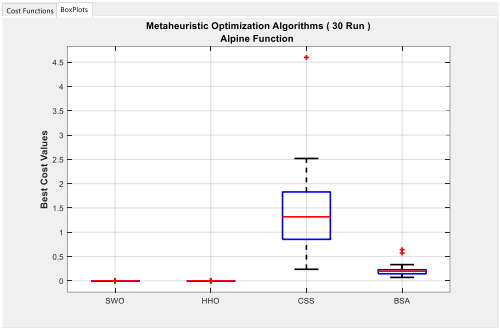
Bu bölümde 50 boyuttaki f_1-f_7 arası test fonksiyonları SWO, HHO, CSS ve BSA algoritmalarıyla 30'ar kez çözülmüş ve elde edilen kutu grafikleri Şekil 28-34 arası ve yakınsama eğrileri ise Şekil 35-41 arasında gösterilmiştir. Tüm algoritmalara göre bulunan sonuçlar en kötü, ortalama, en iyi, standart sapma ve süre olarak Tablo 3'te verilmiştir.



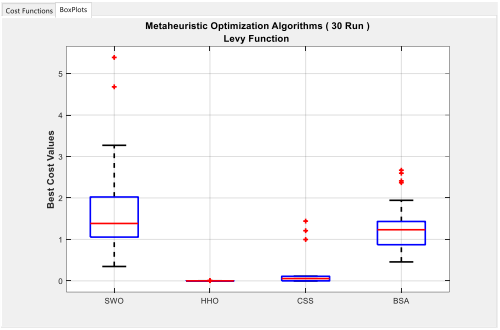
Şekil 28. f_1 için kutu grafiği (50-D)



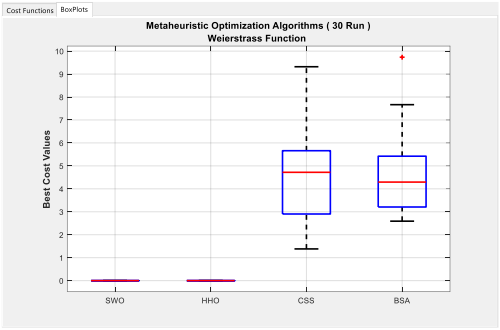
Şekil 29. f_2 için kutu grafiği (50-D)



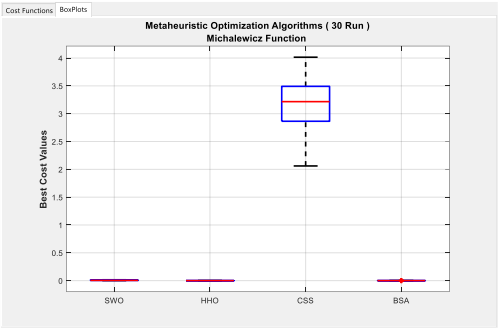
Şekil 30. f_3 için kutu grafiği (50-D)



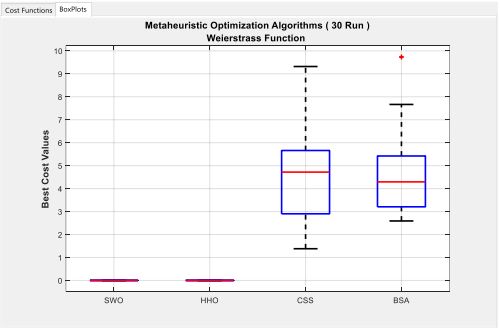
Şekil 31. f_4 için kutu grafiği (50-D)



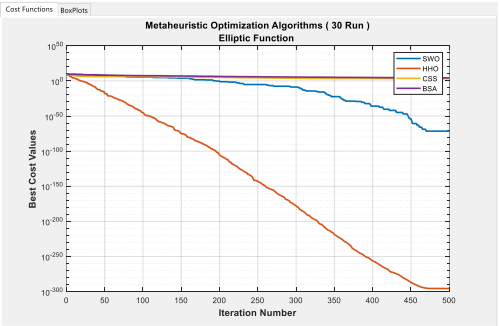
Şekil 32. f_5 için kutu grafiği (50-D)



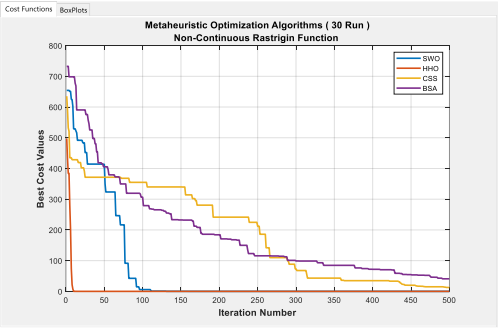
Şekil 33. f_6 için kutu grafiği (50-D)



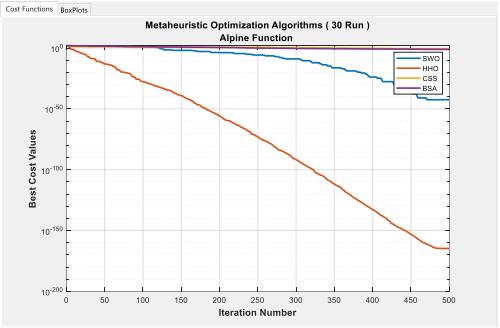
Şekil 34. f_7 için kutu grafiği (50-D)



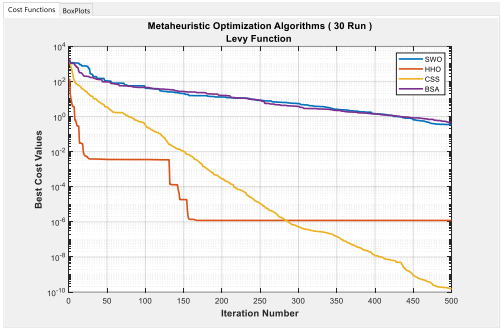
Şekil 35. f_1 için yakınsama eğrileri (50-D)



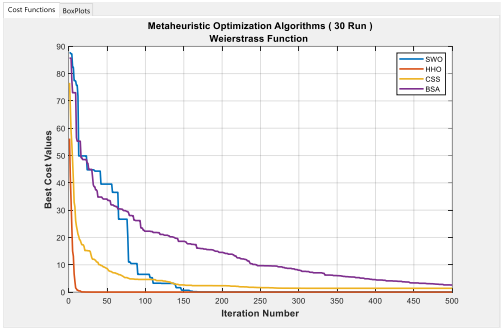
Şekil 36. f_2 için yakınsama eğrileri (50-D)



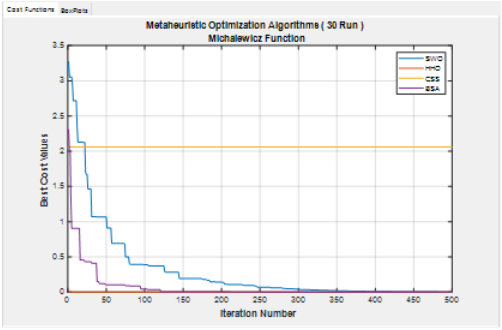
Şekil 37. f_3 için yakınsama eğrileri (50-D)



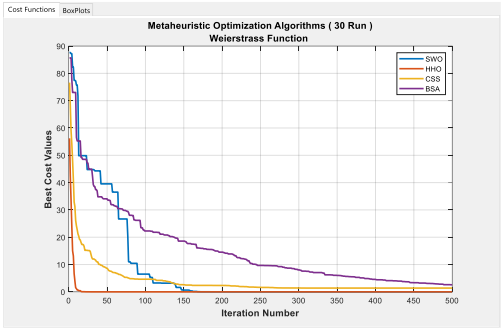
Şekil 38. f_4 için yakınsama eğrileri (50-D)



Şekil 39. f_5 için yakınsama eğrileri (50-D)



Şekil 40. f_6 için yakınsama eğrileri (50-D)



Şekil 41. f_7 için yakınsama eğrileri (50-D)

Şekil 28-41 arası gösterilen 50 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 28-29'da SWO ve HHO algoritmalarının diğerlerine göre iyi bir dağılımlara sahip oldukları ve Şekil 35-36'da ise HHO'nun SWO'ya göre daha hızlı bir şekilde yakınsadıkları görülmektedir. f_3 test fonksiyonu için Şekil 30'da SWO ve HHO algoritmalarının iyi bir dağılımlara sahip oldukları ve Şekil 37'de ise HHO'nun SWO'ya göre daha iyi performans gösterdiği söylenebilir. f_4 test fonksiyonu için Şekil 31'de HHO algoritmasının en iyi dağılıma sahip olduğu ve Şekil 38'de ise CSS algoritmasının HHO ve SWO'ya göre daha iyi performans gösterdiği ifade edilebilir. f_5 test fonksiyonu için Şekil 32'de SWO ve HHO algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 39'da ise HHO algoritmasının SWO ile birlikte iyi yakınsadıkları görülmektedir. f_6 test fonksiyonu için Şekil 33'te SWO, HHO ve BSA algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 40'da ise HHO ile BSA algoritmalarının SWO'ya göre daha iyi yakınsadığı söylenebilir. f_7 test fonksiyonu için Şekil 34'te SWO ile HHO algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 41'de ise HHO algoritmasının SWO'ya göre daha hızlı ve aynı değere yakınsadıkları ifade edilebilir.

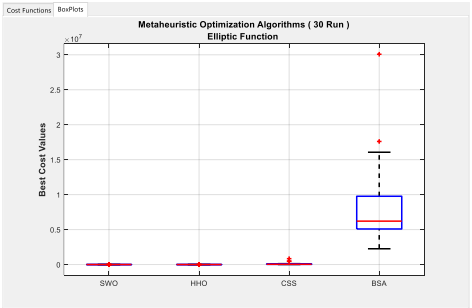
Tablo 3 incelendiğinde SWO algoritmasının f_2 ve f_5'de en iyi çözüm noktalarında, diğer fonksiyonlarda ise HHO ve CSS algoritmalarının en iyi değerleri yakaladıkları görülmektedir. Ortalama süre açısından da SWO algoritmasının en iyi sürede f_1, f_2, f_3, f_4, f_5 *ve* f_6'da olduğu görülmektedir. HHO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

Tablo 3. Algoritmaların 50 Boyutlu test fonksiyonlarına uygulanması durumunda sonuçları (30 bağımsız çalışma)

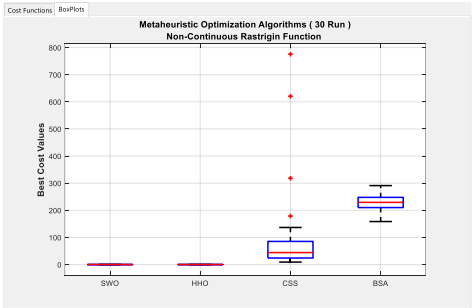
		SWO	*HHO*	*CSS*	*BSA*
f_1	EnKötü	9.571589029e-32	6.41821678e-277	2.871426770e+04	1.996977551e+05
	Ortalama	3.190529677e-33	2.20543306e-278	8.524039799e+03	6.290995202e+04
	EnİyI	2.910475160e-72	2.17773195e-296	1.697173203e+03	2.110301964e+04
	StdSapma	1.718152813e-32	0.000000000e+00	6.795176549e+03	3.691009619e+04
	Süre (s)	0.18087	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)	0.132747	0.438639	0.778613	0.197815
f_3	EnKötü	1.381706423e-25	2.58990167e-156	4.597241239e+00	6.412854319e-01
	Ortalama	9.281336991e-27	2.74891215e-157	1.392081202e+00	2.096120013e-01
	EnİyI	3.146463027e-43	1.71220704e-165	2.383582911e-01	7.118716159e-02
	StdSapma	3.076449348e-26	5.88289395e-157	8.580532196e-01	1.223659873e-01
	Süre (s)	0.112161	0.119756	0.526845	0.131693
f_4	EnKötü	5.388983169e+00	8.679383717e-03	1.441088269e+00	2.668654382e+00
	Ortalama	1.718599612e+00	8.525003795e-04	1.655641668e-01	1.282638371e+00
	EnİyI	3.468654397e-01	1.199906907e-06	1.475388260e-10	4.569776301e-01
	StdSapma	1.128538148e+00	1.792242215e-03	3.586133844e-01	5.898837705e-01
	Süre (s)	0.118881	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)	1.65606	6.3429	3.4381	2.81924
f_6	EnKötü	1.543433394e-02	0.000000000e+00	4.015088642e+00	3.324551946e-05
	Ortalama	5.895320416e-03	0.000000000e+00	3.154592879e+00	4.914669733e-06
	EnİyI	1.451628031e-03	0.000000000e+00	2.060935886e+00	1.497777627e-16
	StdSapma	3.051713844e-03	0.000000000e+00	4.645106288e-01	8.646844487e-06
	Süre (s)	0.272809	1.04624	0.745826	0.327373
f_7	EnKötü	6.670122705e-01	2.497433539e-01	5.248177066e+00	1.326534283e+02
	Ortalama	6.668160079e-01	1.629861398e-01	8.463822177e-01	4.607752485e+01
	EnİyI	6.666946601e-01	7.391894133e-02	6.666703273e-01	2.138019777e+01
	StdSapma	8.387528316e-05	4.287515677e-02	8.193092468e-01	2.134020147e+01
	Süre (s)	0.0927233	0.242556	0.517351	0.078468

6.3. Boyutu 100 Olan Test Fonksiyonları İçin

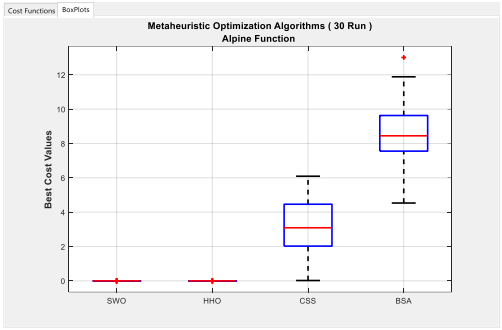
Bu bölümde 100 boyuttaki f_1-f_7 arası test fonksiyonları SWO, HHO, CSS ve BSA algoritmalarıyla 30'ar kez çözülmüş ve elde edilen kutu grafikleri Şekil 42-48 arası ve yakınsama eğrileri ise Şekil 49-55 arasında gösterilmiştir. Tüm algoritmalara göre bulunan sonuçlar en kötü, ortalama, en iyi, standart sapma ve süre olarak Tablo 4'te verilmiştir.



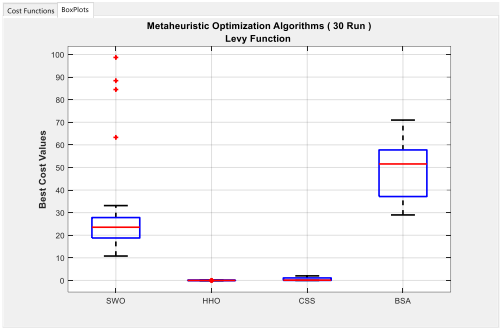
Şekil 42. f_1 için kutu grafiği (100-D)



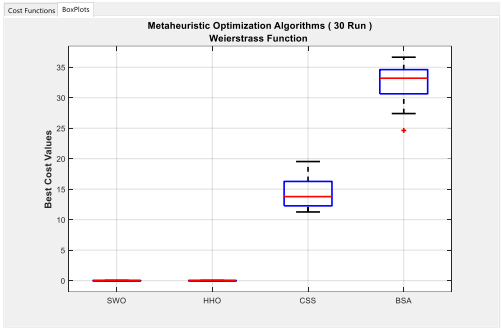
Şekil 43. f_2 için kutu grafiği (100-D)



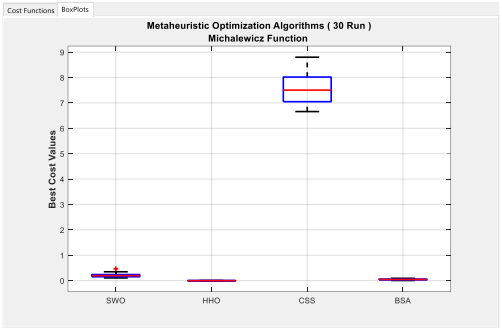
Şekil 44. f_3 için kutu grafiği (100-D)



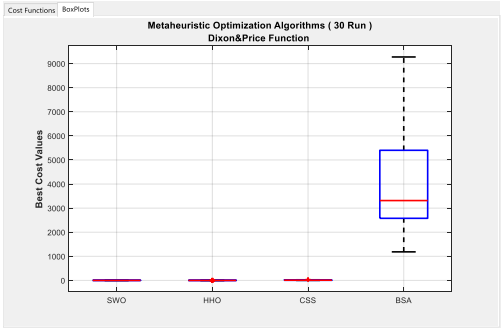
Şekil 45. f_4 için kutu grafiği (100-D)



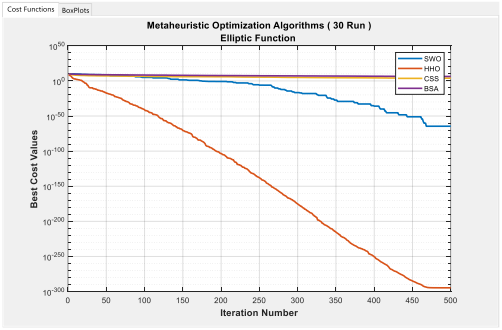
Şekil 46. f_5 için kutu grafiği (100-D)



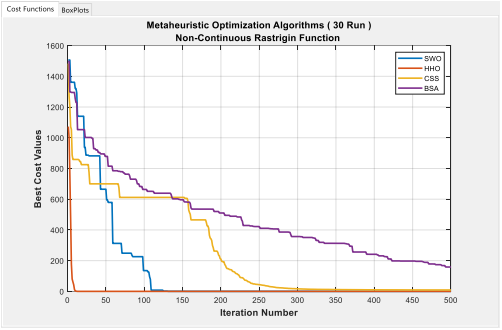
Şekil 47. f_6 için kutu grafiği (100-D)



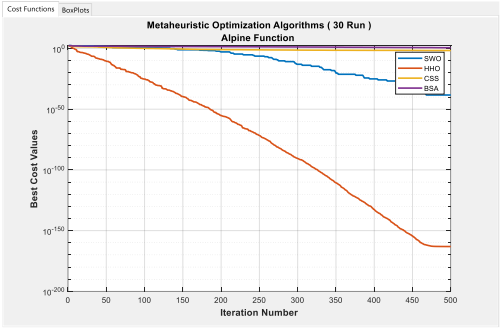
Şekil 48. f_7 için kutu grafiği (100-D)



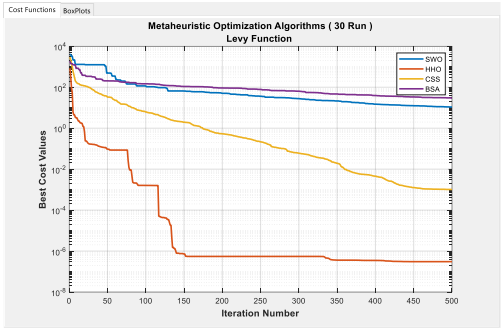
Şekil 49. f_1 için yakınsama eğrileri (100-D)



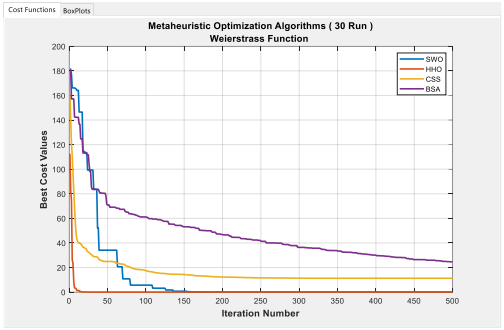
Şekil 50. f_2 için yakınsama eğrileri (100-D)



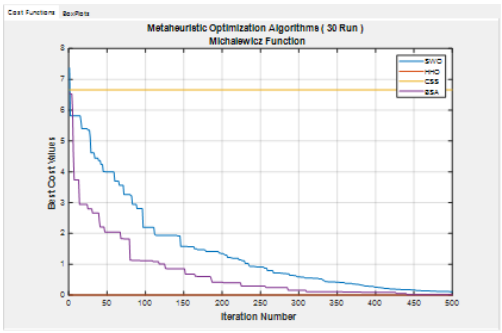
Şekil 51. f_3 için yakınsama eğrileri (100-D)



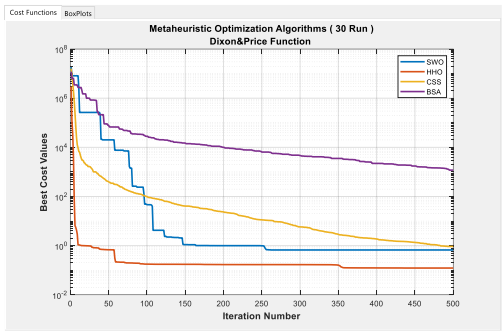
Şekil 52. f_4 için yakınsama eğrileri (100-D)



Şekil 53. f_5 için yakınsama eğrileri (100-D)



Şekil 54. f_6 için yakınsama eğrileri (100-D)



Şekil 55. f_7 için yakınsama eğrileri (100-D)

Şekil 42-55 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 42-43'te SWO ve HHO algoritmalarının diğerlerine göre iyi bir dağılımlara sahip oldukları ve Şekil 49-50'de ise HHO'nun SWO'ya göre daha hızlı bir şekilde yakınsadıkları görülmektedir. f_3 test fonksiyonu için Şekil 44'te SWO ve HHO algoritmalarının iyi bir dağılımlara sahip oldukları ve Şekil 51'de ise HHO'nun SWO'ya göre daha iyi performans gösterdiği söylenebilir. f_4 test fonksiyonu için Şekil 45'de HHO ve CSS algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 52'de ise HHO ve CSS algoritmalarının SWO'ya göre daha iyi performans gösterdiği ifade edilebilir. f_5 test fonksiyonu için Şekil 46'da SWO ve HHO algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 53'te ise HHO algoritmasının SWO ile aynı değere yakınsadıkları görülmektedir. f_6 test fonksiyonu için Şekil 47'de HHO ile BSA algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 54'te ise HHO ve BSA algoritmalarının SWO'ya göre daha iyi yakınsadığı söylenebilir. f_7 test fonksiyonu için Şekil 48'de SWO, HHO ve CSS algoritmalarının iyi bir dağılıma sahip oldukları ve Şekil 55'de ise HHO algoritmasının SWO'ya göre daha hızlı yakınsadığı ifade edilebilir.

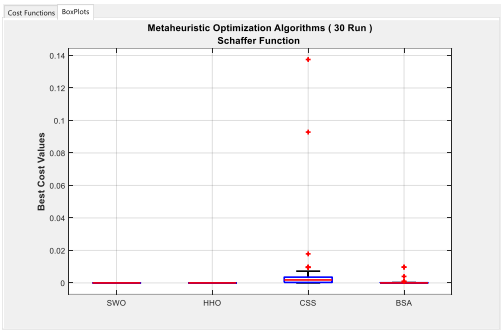
Tablo 4 incelendiğinde SWO algoritmasının f_2 ve f_5'de en iyi çözüm noktalarında, diğer fonksiyonlarda ise HHO algoritmasının en iyi değerleri yakaladıkları görülmektedir. Ortalama süre açısından da SWO algoritmasının en iyi sürede f_1, f_2, f_4, f_5, f_6 *ve* f_7'de olduğu görülmektedir. HHO algoritmasının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

Tablo 4. Algoritmaların 100 Boyutlu test fonksiyonlarına uygulanması durumunda sonuçları (30 bağımsız çalışma)

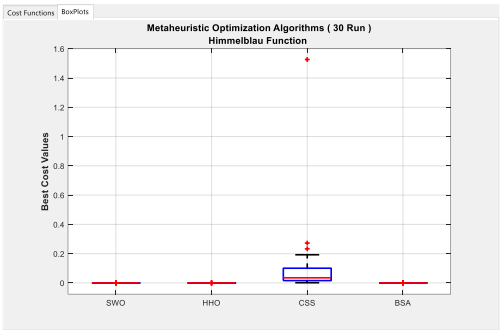
		SWO	*HHO*	*CSS*	*BSA*
f_1	EnKötü	7.499726829e-32	4.94197136e-276	8.392134107e+05	3.008172789e+07
	Ortalama	2.572180203e-33	1.70090182e-277	1.017145069e+05	8.117098571e+06
	Enİyi	2.854394114e-65	2.27855811e-295	8.149396871e+03	2.257152310e+06
	StdSapma	1.345460675e-32	0.000000000e+00	1.773132206e+05	5.553688722e+06
	Süre (s)	0.313932	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)	0.148533	0.348959	0.619033	0.198577
f_3	EnKötü	1.921399255e-26	1.27449386e-154	6.093792619e+00	1.301058367e+01
	Ortalama	1.084071530e-27	7.49256779e-156	3.206623703e+00	8.549302892e+00
	Enİyi	4.724427380e-39	7.94069719e-164	2.101854071e-02	4.532600510e+00
	StdSapma	3.761239860e-27	2.71506276e-155	1.670842189e+00	1.811710678e+00
	Süre (s)	0.125445	0.12431	0.565801	0.121941
f_4	EnKötü	9.869290428e+01	1.654379434e-02	2.015888167e+00	7.097438424e+01
	Ortalama	3.055266038e+01	1.220085645e-03	5.124034157e-01	4.903051232e+01
	Enİyi	1.079862194e+01	3.046131352e-07	1.018482902e-03	2.898540536e+01
	StdSapma	2.191833382e+01	2.940982370e-03	6.374421062e-01	1.207821902e+01
	Süre (s)	0.150351	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)	2.71994	10.1082	5.44946	4.96749
f_6	EnKötü	4.641408978e-01	0.000000000e+00	8.801297790e+00	9.096605556e-02
	Ortalama	2.079836821e-01	0.000000000e+00	7.552437615e+00	4.562120093e-02
	Enİyi	1.056742832e-01	0.000000000e+00	6.659663915e+00	5.905831101e-03
	StdSapma	7.496553776e-02	0.000000000e+00	5.942220006e-01	2.107831378e-02
	Süre (s)	0.341292	1.12568	1.01241	0.581091
f_7	EnKötü	6.679236633e-01	2.508023605e-01	3.567204171e+01	9.275321920e+03
	Ortalama	6.671542799e-01	1.803532231e-01	1.047601157e+01	4.037083487e+03
	Enİyi	6.666794193e-01	1.227066305e-01	9.270521615e-01	1.184919624e+03
	StdSapma	3.562909965e-04	3.725655588e-02	8.010874527e+00	2.091156085e+03
	Süre (s)	0.111163	0.289212	0.651036	0.116422

6.4. Sabit Boyutlu Test Fonksiyonları İçin

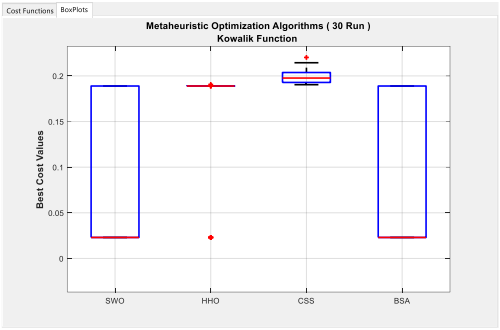
Bu bölümde 2 boyuttaki f_8 ve f_9, 4 boyuttaki f_{10} test fonksiyonları SWO, HHO, CSS ve BSA algoritmalarıyla 30'ar kez çözülmüş ve elde edilen kutu grafikleri Şekil 56-58 arası ve yakınsama eğrileri ise Şekil 59-61 arasında gösterilmiştir. Tüm algoritmalara göre bulunan sonuçlar en kötü, ortalama, en iyi, standart sapma ve süre olarak Tablo 5'te verilmiştir.



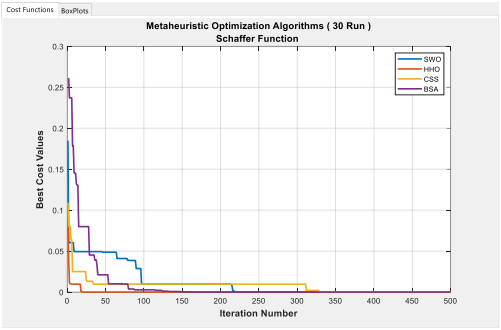
Şekil 56. f_8 için kutu grafiği (2-D)



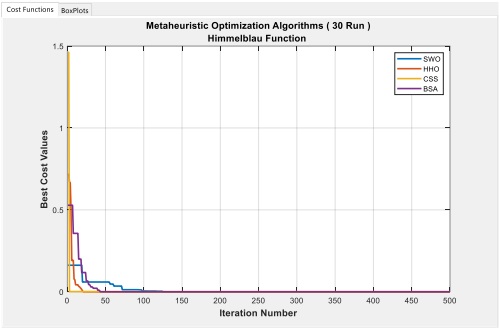
Şekil 57. f9 için kutu grafiği (2-D)



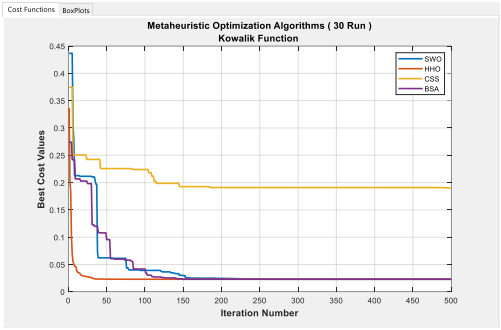
Şekil 58. f10 için kutu grafiği (4-D)



Şekil 59. f8 için yakınsama eğrileri (2-D)



Şekil 60. f9 için yakınsama eğrileri (2-D)



Şekil 61. f10 için yakınsama eğrileri (4-D)

Şekil 56-61 arası gösterilen sabit boyuttaki f_8-f_{10} arası test fonksiyonlarının kutu grafikleri ve yakınsama eğrilerini birlikte inceleyerek aşağıdaki değerlendirmeleri yapabiliriz. f_8 test fonksiyonları için Şekil 56'da SWO ve HHO algoritmalarının diğerlerine göre iyi bir dağılımlara sahip oldukları ve Şekil 59'da ise HHO'nun SWO'ya göre daha hızlı bir şekilde aynı değere yakınsadıkları görülmektedir. f_9 test fonksiyonu için Şekil 57'de SWO, HHO ve BSA algoritmalarının iyi bir dağılımlara sahip oldukları ve Şekil 60'da ise CSS, HHO ve BSA'nın SWO'ya göre daha hızlı ve fakat tüm algoritmaların aynı değere yakınsadıkları söylenebilir. f_{10} test fonksiyonu için Şekil 58'de HHO algoritmasının iyi bir dağılıma sahip olduğu ve Şekil 61'de ise HHO ve BSA algoritmalarının SWO'ya göre daha hızlı oldukları ifade edilebilir.

Tablo 5. Algoritmaların Sabit boyutlu test fonksiyonlarına uygulanması durumunda sonuçları (30 bağımsız çalışma)

		SWO	***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)	0.0716249	0.0820939	0.261233	0.0408193
f_9	EnKötü	1.768068617e-09	1.577721810e-29	1.526359564e+00	1.959677532e-22
	Ortalama	5.998376605e-11	1.367358902e-30	1.134555325e-01	6.589065130e-24
	EnİyI	0.000000000e+00	0.000000000e+00	1.416855131e-03	0.000000000e+00
	StdSapma	3.172261287e-10	3.161573252e-30	2.717440129e-01	3.516722141e-23
	Süre (s)	0.0389045	0.065922	0.271835	0.0298616
f_{10}	EnKötü	1.889337486e-01	1.906626048e-01	2.202368095e-01	1.889337486e-01
	Ortalama	7.830682215e-02	1.558400003e-01	1.993740795e-01	8.383816546e-02
	EnİyI	2.299335416e-02	2.299335416e-02	1.903879040e-01	2.299335416e-02
	StdSapma	7.822504989e-02	6.642365387e-02	7.703616069e-03	7.996574845e-02
	Süre (s)	0.0482818	0.0764176	0.264134	0.0380855

Tablo 5 incelendiğinde SWO algoritmasının f_8 *ve* f_{10}'da en iyi çözüm noktalarında, diğer fonksiyonda ise HHO algoritmasının en iyi değeri yakaladıkları görülmektedir. Ortalama süre açısından da BSA algoritmasının en iyi sürede f_8, f_9 *ve* f_{10}'da olduğu görülmektedir. SWO algoritmasının f_{10}'da ortalama değer açısından diğer algoritmalardan daha iyi standart sapmalar açısından karşılaştırıldığında ise HHO ve CSS algoritmalarından daha kötü bir değere sahip olduğu görülmüştür. HHO algoritmasının en iyi değeri yakaladığı durum için f_8'de ortalama değerler ve standart sapmalar açısından karşılaştırıldığında SWO algoritmasından daha iyi değerlere sahip olduğu görülmektedir.

7. Tartışma

Bu çalışmada SWO algoritmasının performansı hem sabit hem de değişken boyutlu fonksiyonlar üzerinde HHO, CSS ve BSA algoritmalarıyla karşılaştırılmıştır. Elde edilen bulgular, SWO algoritmasının özellikle f_2 (Non-Continuous Rastrigin) ve f_5 (Weierstrass) gibi yüksek sayıda yerel minimum içeren fonksiyonlarda sıfır hata ile sonuçlar üretebildiğini göstermektedir. Bu durum, algoritmanın **yerel minimumlardan kaçınma ve küresel optimuma yakınsamada** etkili olduğunu göstermektedir.

HHO algoritması özellikle yakınsama hızı açısından güçlü bir performans sergilemiş ve SWO'ya göre bazı fonksiyonlarda daha hızlı sonuç üretmiştir. Ancak, HHO'nun bazı durumlarda standart sapma değerlerinin artması, çözüm kararlılığı açısından SWO'ya göre daha az istikrarlı olduğunu düşündürmektedir.

CSS algoritması, fizik tabanlı yapısı sayesinde bazı problemler üzerinde iyi sonuçlar vermiştir (örneğin f_4), ancak genel olarak yüksek hesaplama maliyeti ve süre açısından dezavantajlıdır. BSA ise bazı fonksiyonlarda rekabetçi performans göstermiş olsa da genelde SWO'ya göre daha düşük çözüm kalitesine sahiptir.

Tüm bu bulgular, SWO algoritmasının belirli problem türlerinde oldukça başarılı olduğunu ve uygun parametre ayarları ile daha da iyileştirilebileceğini ortaya koymaktadır. Ancak, bazı durumlarda yakınsama hızının HHO'ya kıyasla daha düşük olması, gelecekte SWO algoritmasında **adaptif parametre ayarı** ve **dinamik strateji geçişi** gibi iyileştirme yönlerinin değerlendirilebileceğini göstermektedir."

8. Sonuçlar

Bu çalışmada doğadaki dişi örümcek arılarının avlanma, yuva yapma ve çiftleşme davranışlarından ilham alan, yani doğadan ilham alan yeni bir sürü tabanlı meta-sezgisel algoritma olan SWO tanıtılarak çeşitli test fonksiyonları üzerinde uygulanmıştır. Bu algoritma örümceği arama, düşen örümceği takip etme ve kaçma, felçli örümceği yuvalama ve dişi örümcek arısının yumurtlamak için çiftleşme davranışı gibi doğadaki dişi örümcek arılarının davranışlarını taklit eden dört aşamadan oluşur. SWO'nun en büyük dezavantajı ise kontrol parametrelerinin performansını maksimuma çıkaracak değerleri bulmaktaki zorluktur.

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

Çalışmada biri unimodal, altı adedi multimodel 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 HHO (1 adet unimodal test fonksiyonu, 3 adet multimodal test fonksiyonu), SWO (2 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 HHO (1 adet unimodal test fonksiyonu, 4 adet multimodal test fonksiyonu) ve SWO (2 adet multimodal test fonksiyonu) olmuştur. Sabit boyut için yapılan değerlendirmede en iyi değerlere sahip algoritmalar sırasıyla SWO (2 adet multimodal test fonksiyonu) ve HHO (1 adet multimodal test fonksiyonu) olmuştur. Çalışmada tüm kutu grafikleri ve yakınsama eğrileri birlikte incelendiğinde SWO ve HHO algoritmalarının en başarılı algoritmalar olduğu görülmektedir. Ayrıca makaledeki Tablo 2, 3, 4, 5'in tümü birlikte incelendiğinde SWO'nun çalışmada kullanılan diğer algoritmalara göre en iyi çözüm süresine sahip olduğu görülmektedir.

Bu çalışma SWO algoritmasının farklı test fonksiyonları üzerindeki performansının araştırılması için bir ön çalışma niteliği taşımaktadır. Bundan sonra yapılacak çalışmalardan biri algoritmanın elektrik elektronik mühendisliği alanında önemli bir yere sahip olan ekonomik güç dağıtımı problemlerine uygulanması olacaktır. Ayrıca yapılacak çalışmalar arasında SWO'nun dezavantajı olan kontrol parametreleri üzerinde durularak algoritmanın performansını yükseltilmesi konusu da yer alabilir.

Referanslar

[1] 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.

[2] S. M. Öztürk and A. Çifci, "A Study in Enhancing Battery Management Systems for Diverse Battery Types," *International Scientific and Vocational Studies Journal*, vol. 7, no. 2, pp. 122-136, 2023. DOI: 10.47897/bilmes.1385510.

[3] T. Aktaş, İ. M. Temel, and A. Saygılı, "Comparative Analysis of Diabetes Diagnosis with Machine Learning Methods," *International Scientific and Vocational Studies Journal*, vol. 8, no. 1, pp. 22-32, 2024. DOI: 10.47897/bilmes.1447878.

[4] A. İ. Çanakoğlu, "Monte Carlo Increased-Radius Floating Random Walk Solution For Potential Problems," *International Scientific and Vocational Studies Journal*, vol. 8, no. 1, pp. 13-21, 2024. DOI: 10.47897/bilmes.1441414.

[5] K. Gencer, G. Gencer, and İ. H. Cizmeci, "Deep Learning Approaches for Retinal Image Classification: A Comparative Study of GoogLeNet and ResNet Architectures," *International Scientific and Vocational Studies Journal*, vol. 8, no. 2, pp. 123-128, 2024. DOI: 10.47897/bilmes.1523768.

[6] U. Saray and U. Çavdar, "Comparison of Different Optimization Algorithms in the Fashion MNIST Dataset," *International Journal of Multidisciplinary Studies and Innovative Technologies (IJMSIT)*, vol. 8, no. 2, pp. 52-58, 2024.

[7] M. D. Demirbaş and D. Çakır (Sofuoğlu), "Evaluation of the Performance of ANN Algorithms with the Bidirectional Functionally Graded Circular Plate Problem," *International Scientific and Vocational Studies Journal (ISVOS)*, vol. 6, no. 2, pp. 103-115, 2022. DOI: 10.47897/bilmes.1207256.

[8] M. Lüy and N. A. Metin, "PID Control Medium Size Wind Turbine Control with Integrated Blade Pitch Angle," *International Scientific and Vocational Studies Journal (ISVOS)*, vol. 6, no. 1, pp. 22-31, 2022. DOI: 10.47897/bilmes.1091968.

[9] B. Durmuş, "Opposite Based Crow Search Algorithm for Solving Optimization Problems," *International Scientific and Vocational Studies Journal (ISVOS)*, vol. 5, no. 2, pp. 164-170, 2021. DOI: 10.47897/bilmes.1031011.

[10] Ö. Yavuz, "An Optimization Focused Machine Learning Approach in Analysing Arts Participative Behavior with Fine Arts Education Considerations," *International Scientific and Vocational Studies Journal (ISVOS)*, vol. 5, no. 2, pp. 241-253, 2021. DOI: 10.47897/bilmes.1029139.

[11] M. D. Demirbaş, M. Oğuz, and İ. Erişen, "Investigation of Buckling Behavior of Beams with Artificial Neural Network," *International Scientific and Vocational Studies Journal (ISVOS)*, vol. 5, no. 1, pp. 94-106, 2021.

[12] E. Dağdevir and M. Tokmakçı, "The Role of Feature Selection in Significant Information Extraction from EEG Signals," *International Scientific and Vocational Studies Journal (ISVOS)*, vol. 5, no. 1, pp. 1-6, 2021.

[13] B. Alızada, "Improved Whale Optimization Algorithm Based On π Number," *International Scientific and Vocational Studies Journal (ISVOS)*, vol. 4, no. 1, pp. 21-30, 2020.

[14] E. Göçmen and O. Derse, "A Tabu Search Algorithm for an Excavator Scheduling Problem," *International Scientific and Vocational Studies Journal (ISVOS)*, vol. 4, no. 1, pp. 1-11, 2020.

[15] M. Taşova, "Effect on the Effective Diffusion and Activation Energy Values of Pea (Pisum sativum L.) Grains of Drying Temperature," *International Scientific and Vocational Studies Journal (ISVOS)*, vol. 3, no. 1, pp. 8-13, 2019.

[16] 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 (ISVOS)*, vol. 2, no. 2, pp. 31-39, 2018.

[17] D. Çakır and M. D. Demirbaş, "Modelling of One-directional Functionally Graded Circular Plates with Artificial Neural Network," *International Scientific and Vocational Studies Journal (ISVOS)*, vol. 3, no. 1, pp. 42-50, 2019.

[18] M. Abdel-Basset, R. Mohamed, M. Jameel, and M. Abouhawwash, "Spider wasp optimizer: a novel meta-heuristic optimization algorithm," *Artificial Intelligence Review*, vol. 56, no. 10, pp. 11675-11738, 2023..

[19] 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.

[20] A. Kaveh and S. Talatahari, "A novel heuristic optimization method: charged system search," *Acta Mechanica*, vol. 213, no. 3, pp. 267-289, 2010.

[21] P. Civicioglu, "Backtracking search optimization algorithm for numerical optimization problems," *Applied Mathematics and Computation*, vol. 219, no. 15, pp. 8121-8144, 2013.

[22] J. Sui, Z. Tian, and Z. Wang, "Multiple strategies improved spider wasp optimization for engineering optimization problem solving," *Scientific Reports*, vol. 14, no. 1, p. 29048, 2024. DOI: 10.1038/s41598-024-78589-8.

[23] İ. Çetinbaş, "Parameter extraction of single, double, and triple-diode photovoltaic models using the weighted leader search algorithm," *Global Challenges (Hoboken, NJ)*, vol. 8, no. 5, p. 2300355, 2024. DOI: 10.1002/gch2.202300355.

[24] H. Liang, W. Hu, L. Wang, K. Gong, Y. Qian, and L. Li, "An Improved Spider Wasp Optimizer for UAV Three-Dimensional Path Planning," *Biomimetics*, vol. 9, no. 12, p. 765, Dec. 2024. DOI: 10.3390/biomimetics9120765.

[25] A. Fathy, "An efficient spider wasp optimizer-based tracker for enhancing the harvested power from thermoelectric generation sources," *Case Studies in Thermal Engineering*, vol. 61, no. 104878, p. 1-26, 2024. DOI: 10.1016/j.csite.2024.104878.

[26] H. Nabil, H. Tayeb, "Effect of chaos on the performance of spider wasp meta-heuristic optimization algorithm for high-dimensional optimization problems," *Mathematical Modelling and Numerical Simulation with Applications*, vol. 5, no. 1, p. 143-171, 2025. DOI: 10.53391/mmnsa.1571964.

[27] D. Evangeline, G. Usha, "Deep Learning Driven LSTM with Spider Wasp Optimizer Algorithm for Frictional Force Based Landslides Prediction Model," *Journal of Intelligent Systems and Internet of Things*, vol. 14, no. 1, p. 293-304, 2025. DOI: 10.54216/JISIoT.140123.

[28] H. Nabil, H. Tayeb, "Effect of chaos on the performance of spider wasp meta-heuristic optimization algorithm for high-dimensional optimization problems," *Mathematical Modelling and Numerical Simulation with Applications*, vol. 5, no. 1, p. 143-171, 2025. DOI: 10.53391/mmnsa.1571964.

[29] Y. Gao, Z. Li, H. Wang, Y. Hu, H. Jiang, X. Jiang, D. Chen, "An Improved Spider-Wasp Optimizer for Obstacle Avoidance Path Planning in Mobile Robots," *Mathematics*, vol. 12, no. 2604, p. 1-25, 2024. DOI: 10.3390/math12172604

[30] S. Saber, S. Salem, "High-Performance Technique for Estimating the Unknown 1Parameters of Photovoltaic Cells and Modules Based on 2Improved Spider Wasp Optimizer," *Sustainable Machine Intelligence Journal*, vol. 5, p. 1-14, 2023. DOI: 10.61185/SMIJ.2023.55102.

[31] https://benchmarkfcns.info/fcns (Access Date: 10.03.2025).

[32] https://al-roomi.org/benchmarks (Access Date: 10.03.2025).