KOCAELİ ÜNİVERSİTESİ * FEN BİLİMLERİ ENSTİTÜSÜ

PARÇACIK SÜRÜSÜ OPTİMİZASYON ALGORİTMASI İLE BULANIK – NÖRAL AĞ EĞİTİMİ

YÜKSEK LİSANS

Elektronik ve Haberleşme Müh. Seçkin TAMER

Anabilim Dalı: Elektronik ve Haberleşme Mühendisliği

Danışman : Yrd. Doç. Dr. Cihan KARAKUZU

KOCAELİ, 2007

KOCAELİ ÜNİVERSİTESİ * FEN BİLİMLERİ ENSTİTÜSÜ

PARÇACIK SÜRÜSÜ OPTİMİZASYON ALGORİTMASI İLE BULANIK - NÖRAL AĞ EĞİTİMİ

YÜKSEK LİSANS TEZİ

Elektronik ve Haberleşme Müh. Seçkin TAMER

Tezin Enstitüye Verildiği Tarih: 04 Haziran 2007

Tezin Savunulduğu Tarih: 26 Haziran 2007

Tez Danışmanı	Üye	Üye
Yrd. Doç. Dr. Cihan KARAKUZU	Prof. Dr. Tülay Yıldırım	Doc. Dr. Yaşar Becerikli
(.....)	(.....)	(.....)

KOCAELİ, 2007

ÖNSÖZ VE TEŞEKKÜR

Günümüzde biyolojik sistemlerden esinlenilerek ortaya çıkarılmış bir çok yöntem bilimsel hesaplama alanında kullanılmaktadır. Örneğin, yapay sinir ağları; insan beyninin basitleştirilmiş bir modelidir. Bulanık mantık; insanın düşünce sistemini örnek alan bir karar verme sistemidir. Genetik algoritmalar; insan evriminden, parçacık sürüsü algoritması ise kuş sürülerinden esinlenilerek ortaya çıkarılmıştır. Doğadaki kusursuz düzenden esinlenilerek ortaya konmuş bu yöntemlere verilebilecek örneklerin sayısını arttırmak mümkündür.

Bulanık mantık işleyişinin, bir yapay sinir ağı içerisinde gerçekleştirildiği Neuro – Fuzzy (bulanık – nöral) ağlar günümüzde başta kontrol olmak üzere bir çok alanda kullanılmaktadır. Bu ağlar; kontrolör olarak kullanılacağında, bir çok sistemde olduğu gibi, bir en iyilemeye (optimizasyon) ihtiyaç duyarlar. Genel anlamda optimizasyon; bir işin en doğru, en verimli şekilde yapılmasıdır ve bir çok yöntemle yapılabilir. Bu tez çalışmasında "Parçacık Sürüsü Optimizasyon Algoritması" kullanılarak bulanık-nöral ağın optimizasyonu gerçekleştirilmiştir. Yapılan çalışmanın, bu alanda yapılan çalışmalara katkıda bulunmasını dilerim.

Beni bu alanda çalışmaya yönlendiren, her türlü yardım ve teşvikte bulunan tez danışmanım Yrd. Doc. Dr. Cihan KARAKUZU'ya, ve hayatımın her aşamasında maddi ve manevi destekleriyle beni ayakta tutup, bu günlere getiren sevgili aileme sonsuz teşekkürü bir borç bilirim.

Seçkin TAMER
Haziran 2007, KOCAELİ

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

SİMGELER DİZİNİ VE KISALTMALAR

P_s: Sürüdeki parçacık sayısı
x_{nxD}: D adet parametre içeren n adet parçacığı temsil eden matris
x_i: Sürüdeki i'ninci parcaçık
$pbest_i$: i'ninci parçacık için pbest (parçacığın kendisinin şu ana kadar ki en iyi değeri bulduğu konum) değeri
gbest: Sürüdeki tüm parçacıklar için şu ana kadar elde edilen en iyi konum
v_i: i'ninci parçacığın hızı
c_1, c_2: öğrenme faktörleri
w: eylemsizlik ağırlığı
p^*_{ij}: i'ninci parçacığın j'ninci parametresi için ortalama en iyi değer (Geliştirilmiş bir PSO yöntemi için)
J: Ölçüt (cost) fonksiyonu, uygunluk değeri, performans indeksi
y: gerçek çıkış
y_d: istenilen çıkış
e: hata
P_c: çaprazlama olasılığı
P_m: mutasyon olasılığı
w_{ij}: Çok katmanlı yapay sinir ağındaki i'ninci girişten j'ninci hücreye bağlantı ağırlığı
b_{ij}: i'ninci katmandaki j'ninci hücrenin eşik değeri (bias)
$O_{l,i}$: l'ninci katmandaki i'ninci hücrenin çıkışı (ANFIS için)
μ_A: Girişin A bulanık kümesine üyelik derecesi
c_i: Gauss üyelik fonksiyonunun merkezi
σ_i: Gauss üyelik fonksiyonunun varyansı
Π: Cebirsel çarpım (T-norm) işlevini gerçekleştiren düğüm
w_i: i'ninci kuralın aktiflik derecesi (kural ateşleme seviyesi) (ANFIS için)
$\overline{w_i}$: i'ninci kuralın normalize aktiflik derecesi (ANFIS için)
f_i: i'ninci kural çıkışı (ANFIS için)p_i, q_i, r_i: i'ninci kural için 1. derece sugeno katsayıları (ANFIS için)
$\sum$: Toplama işlemini gerçekleştiren düğüm
z: Kuralların ağırlıklı ortalamasıyla elde edilen çıkış
s_i: sigmoidal fonksiyonun eğimi
c_i: sigmoidal fonksiyonunun merkezi
R_i: i'ninci kural aktiflik derecesi (TRFN için)
h_i: i'ninci kural için geri besleme sinyali (TRFN için)
w_{ij}: j'ninci kuraldan, i'ninci kuralın geri beslemesine bağlantı ağırlığı (TRFN için)
a_{ij}: i'ninci kuralın çıkışının hesaplanması için kullanılan j'ninci parametre (TRFN için)
f_i: i'ninci kural çıkışı (TRFN için)
N_r: Kural sayısı
u_{ij}: i'ninci girişin, j'ninci bulanık kümesinin girişini (RFNN2 için)

O_{ij}: i'ninci girişin, j'ninci bulanık kümesinin çıkışını (RFNN2 için)
θ_{ij}: i'ninci girişin, j'ninci bulanık kümesi için geri besleme sinyali bağlantı ağırlığı (RFNN2 için)
w_{ij}: i'ninci kuraldan, j'ninci çıkışa bağlantı ağırlığı (RFNN2 için)
y_{set}, y_{ref}, y_r: Sistem çıkışında elde etmek istediğimiz değer (ön ayar değeri)
y_p: Sistem çıkışında elde edilen değer
u: Kontrolörde üretilen ve sisteme uygulanan kontrol sinyali
J: DC motor için rotorun eylemsizlik momenti ($kg.m^2/s^2$)
b: mekanik sistemin sönümleme sabiti (Nms)
T: Motorun torku (Nm)
K, K_e, K_t: DC motor için sabitler (Nm/A)
R: Stator direnci (ohm)
L: İndüktans (H)
i: Armatür akımı (A)
V: Motora uygulanan gerilim (V)
Θ: Motorun açısal konumu

Kısaltmalar

PSO: Parçacık Sürüsü Optimizasyonu
GA: Genetik Algoritma
LM: Levenberg-Marquardt
FPCG: Fletcher-Powell Conjugate Gradient
PRCG: Polak-Ribiére Conjugate Gradient
GD: Gradient Descent (Eğim iniş yöntemi)
LSE: Least Square Error (En küçük karesel hata)
HPSOGA: Hibrit Parçacık Sürüsü Optimizasyonu Genetik Algoritma
ANFIS: Adaptive-Network-Based Fuzzy Inference System (Uyarlamalı ağ tabanlı bulanık çıkarım sistemi)
TSK: Takagi-Sugeno-Kang Bulanık çıkarım yöntemi
TRFN: TSK type Recurrent Fuzzy Network (TSK türü yinelemeli bulanık ağ)
RFNN: Recurrent Fuzzy Neural Network (Yinelemeli bulanık nöral/sinirsel ağ)
FPGA: Field Programmable Gate Array (Alan Programlanabilir Kapı Dizisi)

PARÇACIK SÜRÜSÜ OPTİMİZASYON ALGORİTMASI İLE BULANIK – NÖRAL AĞ EĞİTİMİ

Seçkin TAMER

Anahtar Kelimeler: Parçacık Sürüsü Optimizasyonu, Genetik Algoritma, Bulanık-Nöral Ağ Eğitimi, Kontrolör Parametrelerinin Optimizasyonu

Özet: Günümüzde, biyolojik sistemlerden esinlenilmiş bir çok hesaplama tekniği bulunmaktadır. Bu tez çalışmasının temelinde de kuş sürülerinin davranışlarından esinlenilerek ortaya çıkarılmış bir yöntem olan "Parçacık Sürüsü Optimizasyon (PSO) Algoritması" bulunmaktadır. Bu tarz yöntemlerin, geriye yayılım gibi klasik yöntemlere karşı bir çok üstünlüğü vardır. Bu yöntemler, çok parametreli sistemlerin optimizasyonunda, diferansiyel denklem çözümü ve gradyen hesaplanması gerektirmediği için işlem yükünü ve işlem zamanını azaltmaktadır.

Bu tez çalışması içerisinde, ilk olarak PSO algoritması anlatılmıştır. Sonrasında algoritmanın başarımını artırmak için ortaya konan karma (hibrit) algoritma anlatılmıştır. Karma algoritmada; PSO algoritması içersine, genetik algoritmaya özgü çaprazlama ve değişim operatörleri dahil edilmiştir. Yapılan benzetim sonuçlarından, algoritmanın başarımının önemli ölçüde arttığı görülmüştür.

Bulanık-nöral ağ parametrelerinin belirlenmesi çok parametreli bir optimizasyon problemidir. Bu tez çalışmasında, bu problemin çözümü, PSO algoritması kullanılarak yapılmıştır. Farklı yapılardaki bulanık-nöral ağların eğitilmesi ve iki örnek sistem üzerindeki kontrol başarımları sunulmuştur. İlk olarak ayrık zamanda tanımlanmış tek giriş-tek çıkışlı dinamik bir sistemi kontrol edebilmek için yinelemeli bulanık-nöral ağ, tez içersinde anlatılan iki algoritma ile eğitilmiştir. Benzetim sonuçlarına göre algoritmaların karşılaştırması yapılmış ve karma algoritmanın üstünlüğü görülmüştür.

Tez içersinde son çalışmada ise üç farklı yapıdaki bulanık-nöral ağ, karma algoritma ile DC motor hızını kontrol etmek için eğitilmiştir. Buradaki amaç farklı yapılardaki bulanık-nöral ağların, sürekli zamanda tanımlanmış örnek bir sistem üzerindeki başarımlarını karşılaştırmaktır. Yapılan benzetimlerde karma algoritma ile eğitilen bulanık-nöral ağların oldukça iyi kontrol sonuçları verdiği görülmüştür.

FUZZY NEURAL NETWORK LEARNING USING PARTICLE SWARM OPTIMIZATION ALGORITHM

Seçkin TAMER

Key Words: Particle Swarm Optimization, Genetics Algorithm, Neuro – Fuzzy Network Training, Optimization of Controller's parameters

Abstract: Nowadays; there are a lot of computation techniques inspired by biological systems. This thesis's basis is based on "Particle swarm optimization (PSO) algorithm" inspired by the behavior of bird flocks. These computation techniques have more superiority than classical techniques like back propagation. These techniques don't need the solution of differential equation and partial gradient, since they decrease the computation time and complexity for purposes of multi parameter optimization problem solutions.

In this thesis, firstly PSO algorithm is explained and then the concept of hybrid algorithm, which is proposed to increase the success of PSO algorithm. In hybrid algorithm; crossover and mutation operators, which belongs to genetics algorithm, are inserted in PSO algorithm. As can be seen from the simulation results, it's shown that hybrid algorithm's performance is superior that of PSO.

Determination of fuzzy neural network's (FNN) parameters is a multi parameter optimization problem. In this dissertation, this problem is solved by using PSO algorithm and training of the different FNN controller architectures and their performances is presented based on two sample systems. Firstly recurrent FNN is trained by two algorithms explained in this thesis to control the single input-single output dynamical plant defined in discrete time domain. For this plant, comparison of classical PSO and hybrid PSO algorithms is made and it is shown superiority of hybrid algorithms with numerous simulation results.

Finally, 3 different types of FNN are trained by hybrid algorithm to control the speed of DC motor. And then the performances of FNN's are compared on the simulation system defined in continuous time domain. As a result, it is shown that the FNN trained by hybrid algorithm gives successful control results.

BÖLÜM 1: GİRİŞ

Doğadaki kusursuz olarak işleyen düzen, tarih boyunca insanlara esin kaynağı olmuştur. Gündelik hayatımızı kolaylaştıran bir çok buluş doğadaki sistemlerden esinlenilerek icat edilmiştir. Bilimsel hesaplama alanında kullanılan bir çok yöntemde biyolojik sistemlerden esinlenilerek ortaya konulmuştur. Örneğin, yapay sinir ağları; insan beyninin basitleştirilmiş bir modelidir. Bulanık mantık; insanın düşünce sistemini örnek alan bir karar verme sistemidir. Genetik algoritma, karınca koloni algoritması, yapay bağışıklık algoritması gibi bir çok yöntem biyolojik sistemlerden esinlenilerek ortaya çıkarılmıştır. Parçacık sürüsü algoritması da kuş sürülerinin davranışlarından esinlenilerek ortaya çıkarılmıştır. Algoritmanın geliştirilmesiyle ilgili literatürdeki önemli çalışmalar aşağıda sunulmuştur.

Parçacık sürüsü optimizasyonu (PSO) ilk olarak; 1995 yılında J.Kennedy ve R.C.Eberhart tarafından "Particle Swarm Optimization" isimli çalışmalarıyla IEEE uluslararası neural network konferansında duyurulmuştur [1]. Kennedy ve Eberhart özellikle biyolog Frank Heppner'in geliştirdiği kuş sürülerinin davranışıyla ilgili modelle ilgilenmişlerdir. Bu modelde kuşların yiyecek aramaları ve yiyecek bulunduktan sonra kuşların sürü halinde yiyeceğe doğru yönelmeleri modellenmiştir. Bu modeldeki en önemli nokta yiyeceği ilk bulan kuşun, diğerlerine rehberlik etmesi ve bireyler arasındaki sosyal bilgi paylaşımıdır. Kennedy ve Eberhart'ın bu çalışmasında; algoritmanın şimdiki durumunu almadan önceki denemeleri ve ortaya atılan çeşitli yöntemler açıklanmış, son olarak algoritmanın güncel hali verilmiştir.

1998 yılında R. Eberhart ve Yuhui Shi tarafından "A Modified Particle Swarm Optimizer" isimli çalışmalarıyla gelişmiş PSO yöntemi ortaya atılmıştır [2]. Bu yöntemde çözümü yakınsama üzerinde büyük etkisi olan "eylemsizlik ağırlığı" isimli parametre algoritmaya dahil edilmiştir.

Yine 1998 yılında R. Eberhart ve Yuhui Shi tarafından yayınlanan bir çalışmada, algoritma içerisindeki bazı parametrelerin alması gereken değerlerle ilgili örnek bir sistem üzerinde elde edilen sonuçlar sunulmuştur. Parametrelerin alması gereken tipik değerleri verilmiştir. Uygun değerler seçilmesi halinde algoritmanın başarımının arttığı görülmüştür [3]. Daha sonra 2004 yılında Zhang Li-ping, Yu Huan-jun ve Hu Shang-xu tarafından benzer bir çalışma yayınlanmıştır [4].

Algoritmanın başarımını arttırmak için yapılan çalışmalar günümüzde de devam etmektedir. Çeşitli araştırmacılar tarafından yapılan çalışmalar da, hibrit (karma) algoritma yapıları üzerinde durulmaktadır. Bu yapılarda çeşitli algoritmaların üstün olan yönleri bir araya getirilmekte ve böylece algoritmanın daha iyi sonuçlar vereceği düşünülmektedir. Literatürde bu yönde yapılan çalışmalardan bazıları aşağıda sunulmuştur.

2004 yılında Chia-Feng Juang tarafından yapılan çalışmalar sonucunda Genetik algoritma ile PSO algoritmasının birleşiminden elde edilen ve HPSOGA olarak adlandırılan hibrit bir algoritma sunulmuştur [5]. Bu tez çalışmasında da bu yöntem üzerinde durulmuş, bu yöntemle bulanık-nöral kontrolör eğitimi gerçekleştirilmiştir.

Yine 2004 yılında uluslararası Güç Sistemleri Konferansında sunulan bir çalışmada önerilen yöntemle parçacıklar güncellenirken, rasgele seçilen bir parçacığın konumu da dikkate alınmış, böylece bireyler arası bilgi paylaşımının artması sağlanmış ve algoritmanın yerel çözümlere takılması engellenmeye çalışılmış [6].

2006 yılında Amerika kontrol konferansında; M. Aliyari Shoorehdeli, M. Teshnehlab, A. K. Sedigh, tarafından bir çalışma sunulmuştur [7]. Bu çalışmada iki giriş, tek çıkışlı sistemi modelleyebilmek için ANFIS yapısındaki ağın giriş parametreleri PSO algoritması ile, sonuç parametreleri ise Gradient Descent yöntemiyle bulunmuştur.

Yine 2006 yılında A. A. A. Emsin ve G. Lambert-Torres tarafından yayınlanan bir çalışmada ortaya konan yöntemde; mutasyon olarak adlandırılan bir işlemle, sürüdeki parçacıklardan rasgele seçilen birinin yeri değiştiriliyor. Bir araya toplanan

parçacıklardan birinin, topluluktan ayrılmasıyla, algoritmanın yerel çözümlere takılması engellenmeye çalışılmıştır [8].

Parçacık sürüsü algoritması; günümüzde, fonksiyon optimizasyonu, bulanık sistem kontrolü, yapay sinir ağı eğitimi gibi bir çok alanda başarıyla uygulanabilmektedir. Algoritmanın kullanımıyla ilgili literatürden seçilen bazı çalışmalar aşağıda verilmiştir.

2005 yılında yayınlanan bir çalışmada, yazarlar 3 farklı problemin çözümü için, 3 farklı yapıdaki yapay sinir ağını PSO ile eğitmişler ve elde ettikleri sonucu, ağ eğitiminde kullanılan klasik geriye yayılım algoritmasıyla yapılan eğitim sonuçlarıyla karşılaştırmışlardır [9].

2005 yılında C. F. Juang ve Chao-Hsin Hsu tarafından yayınlanan çalışmada, aynı yazara ait [5]'te verilen TSK biçimli yinelemeli bulanık ağ ile ters sistem yapısında kontroller tasarlanmış. Ağ eğitiminde simplex metodu ile PSO dan oluşan hibrit bir yapı kullanılmış. FPGA üzerinde bu kontrolör gerçeklenmiş ve su sıcaklığı kontrol edilmiştir [10].

2006 yılında, yine C. F. Juang, C. F. Lu tarafından yayınlanan bir başka çalışmada ise elektriksel bir sistem, PI kontrolör ile kontrol edilmek istenmiş, gerekli olan K_p ve K_i katsayıları bulanık denetleyici tarafından üretilmiştir. Bulanık denetleyiciye ait üyelik fonksiyonu parametreleri ise aynı yazara ait, [5] te verilen karma algoritma ile belirlenmiştir [11].

Bu tez içinde yapılan çalışmalar şu düzen içinde anlatılmıştır: Bölüm 2'de, Parçacık sürüsü optimizasyon algoritması anlatılmıştır. Genel olarak sürü kavramı ve algoritmanın sosyal davranışlarla ilişkisi anlatılmıştır. Algoritmanın yapısı, çeşitli türleri, güçlü ve zayıf yönleri anlatılmıştır. Algoritmanın işleyişini daha iyi kavramak için gerçekleştirilen benzetim çalışmalarından elde edilen sonuçlar sunulmuştur.

Bölüm 3'te, kısaca Genetik algoritmadan bahsedilmiştir. Parçacık sürüsü algoritmasının başarımını artırmak için algoritma içersine ilave edilen Genetik algoritmaya ait operatörler ve karma arama algoritmasının işleyişi ayrıntılı olarak ele alınmıştır.

Bölüm 4'te, genel olarak bulanık-nöral ağlar anlatılmıştır. Öncelikle adaptif bulanık çıkarım sistemleri için temel olan ANFIS ağ yapısı tanıtılmış, sonrasında da 2 adet yinelemeli ağ yapısı üzerinde durulmuştur. Ağ yapıları, katmanları ve ağda bulunan parametreler ayrıntılı olarak tanıtılmıştır.

Bölüm 5'te, bulanık- nöral ağların kontrolör olarak eğitilmesi üzerinde durulmuş ve iki örnek sistem üzerinde kontrol benzetimleri yapılmıştır. İlk örnek olarak dördüncü bölümde anlatılan yinelemeli bulanık-nöral ağın, tek giriş-tek çıkışlı dinamik bir sistemi istenilen ön ayar değerinde tutabilmek için, kontrolör olarak eğitilmesi anlatılmıştır. Eğitim; PSO algoritması ve karma arama algoritması ile gerçekleştirilmiş, elde edilen sonuçlarla iki algoritmanın karşılaştırılması yapılmıştır. İkinci örnek olarak ise dördüncü bölümde anlatılan üç farklı yapıdaki bulanık-nöral ağ, karma arama algoritması kullanılarak kontrolör olarak eğitilmiş ve DC motorun hızını kontrol etmek için benzetimler yapılmıştır. Elde edilen sonuçlarla ağ yapılarının başarımları karşılaştırılmıştır.

Bölüm 6'de ise, bu tez çalışmasında elde edilen sonuçların genel bir değerlendirmesi yapılmış, öneri ve sonuçlar verilmiştir.

BÖLÜM 2: PARÇACIK SÜRÜSÜ OPTİMİZASYON ALGORİTMASI

2.1 Giriş

Bu bölümde sürü davranışlarından esinlenilerek ortaya çıkarılmış en yeni optimizasyon tekniklerinden birisi olan Parçacık Sürüsü (particle swarm) Optimizasyon (PSO) Algoritması anlatılmıştır. PSO; 1995 yılında J.Kennedy ve R.C.Eberhart tarafından; kuş sürülerinin davranışlarından esinlenilerek ortaya çıkarılmış populasyon tabanlı optimizasyon tekniğidir [1]. Algoritmayı uygulamak oldukça basittir ve çok parametreli, doğrusal olmayan problemlerin çözümü için kullanılmaktadır. Bu bölümde sırasıyla; sürü zekası, algoritmanın sosyal yaşamla ilişkisi, yapısı ve farklı türleri anlatılmıştır. Son kısımda ise yapılan benzetim çalışmalarıyla algoritmanın başarımı anlatılmıştır.

2.2 Sürü Zekası

Günümüzde biyolojik sistemlerden esinlenilerek ortaya çıkarılmış birçok yöntem hesaplama problemlerinin çözümünde kullanılmaktadır. Örneğin yapay sinir ağları insan beyninin basitleştirilmiş bir modelidir, genetik algoritma ise insan evriminden esinlenilerek ortaya çıkarılmıştır. Biyolojik sistemlerin başka bir çeşidi olan sosyal sistemler, özellikle bireyin çevresiyle ve diğer bireylerle olan etkileşimi ve kollektif (ortak) davranışları incelemektedir. Bu davranışlar sürü zekası olarak adlandırılmaktadırlar. Sürü zekası (Swarm Intelligence); yetenekleri kısıtlı canlıların bir takım haline geldiklerinde üstün yetenek ve yönetim gerektiren davranışlar sergilemesidir. Karıncaların yiyeceğe giden en kısa yolu bulmaları, kuşların sürü halindeyken ki uçuş şekilleri ve yine karıncaların aralarında görev paylaşımı yapmaları ve kendilerini örgütleyebilmeleri sürü zekasına örnek olarak gösterilebilir. Sürü zekasının en önemli özelliklerinden birisi merkezi bir yönetimin olmamasıdır (decentralization). Sürüde yer alan bireylerin çok basit karar mekanizmaları vardır ve

sadece o işi yapmakla kendilerini yükümlü kılarlar. Her etmenin kendi işini yapması sonucu ortaya ancak karmaşık bir karar mekanizmasına sahip bir yöneticinin vereceği kararla gerçekleşecek eylemler çıkar. Sürü zekasının diğer bir özelliği de kendini örgütleyebilen(selforganisation) bireylerden oluşmasıdır. Sürü bireyleri, değişen şartlara göre kendilerini örgütleyebilirler, yaptıkları işin cinsini değiştirme ve duruma göre kendilerine uygun işi tayin edebilme yeteneklerine sahiptirler. Mesela yiyecek taşıyan bir karınca, taşınacak yiyecekler sona erince kendisini bir sonraki yapılacak iş olan çevre temizliği için görevlendirebilir. Bu özellik sürülerde görev paylaşımının gerçekleşmesini sağlar. Aşağıda sürü zekasına ait bazı özellikler verilmiştir.

• Kendi aralarında lokal iletişime sahip çok sayıda bireyden oluşur.
• Otonom bir sistemdir tüm bireyler kendi kontrolünü yaparlar.
• Ortak bir amaç doğrultusunda bir arada bulunurlar.
• Tüm bireyler birbirlerinin davranışlarından etkilenirler.
• Merkezi bir sistem tarafından yönetilmezler.
• Esnek bir yapıları vardır. Meydana gelen olumsuz durumlarda pozisyon değiştirebilir ve yeni ortama kolaylıkla ayak uydurabilirler.
• Bireylerin çevre ile iletişimleri vardır. Ortam bilgilerini elde edebilirler.
• Birimlerden bazıları çalışmaz hale gelse bile sistem devam ettirilebilir.

Hesaplama alanında kullanılan ve sürülerden esinlenilerek ortaya konulan iki popular metod vardır. Bunlar "karınca koloni optimizasyonu"(KKO) ve "parçacık sürüsü optimizasyonu"(PSO) dur. KKO karıncaların davranışlarından esinlenilerek ortaya konmuş bir yöntemdir ve ayrık optimizasyon problemlerinde başarılı uygulamaları vardır. PSO kavramı esasında sosyal yaşamın basitleştirilmiş bir benzetimidir. Daha sonra bu benzetim bir optimizasyon yöntemi olarak kullanılmaya başlanmıştır [12].

2.3 Parçacık Sürüsü Algoritması

Parçacık Sürüsü (particle swarm) Optimizasyonu (PSO); 1995 yılında J.Kennedy ve R.C.Eberhart tarafından; kuş sürülerinin davranışlarından esinlenilerek geliştirilmiş populasyon tabanlı rasgele arama algoritmasıdır. Doğrusal olmayan problemlerin çözümü için tasarlanmıştır. Çok parametreli ve çok değişkenli optimizasyon

problemlerine çözüm bulmak için kullanılmaktadır. PSO, genetik algoritmalar gibi evrimsel hesaplama teknikleriyle bir çok benzerlik gösterir. Sistem rasgele çözümler içeren bir populasyonla başlatılır ve nesilleri güncelleyerek en optimum çözümü araştırır. PSO da parçacık olarak adlandırılan olası muhtemel çözümler, o andaki optimum parçacığı izleyerek problem uzayında dolaşırlar. PSO'nun klasik optimizasyon tekniklerinden en önemli farklılığı türev bilgisine ihtiyaç duymamasıdır. Bu özellik bir çok problemin çözümü için gerekli olan karmaşık işlem yükünün hafifletilmesini sağlamaktadır. PSO'yu uygulamak, algoritmasında ayarlanması gereken parametre sayısının az olması sebebiyle oldukça basittir. PSO; fonksiyon optimizasyonu, bulanık sistem kontrolü, yapay sinir ağı eğitimi gibi bir çok alanda başarıyla uygulanabilmektedir [9-11, 13-15].

Yukarıda da bahsedildiği gibi PSO kuş sürülerinin davranışlarının bir benzetimidir. Kuşların uzayda, yerini bilmedikleri yiyeceği aramaları, bir probleme çözüm aramaya benzetilir. Kuşlar yiyecek ararken yiyeceğe en yakın olan kuşu takip ederler. Parçacık olarak adlandırılan her tekil çözüm, arama uzayındaki bir kuştur. Parçacık hareket ettiğinde, kendi koordinatlarını bir fonksiyona gönderir ve böylece parçacığın uygunluk değeri ölçülmüş olur. (Yani yiyeceğe ne kadar uzaklıkta olduğu ölçülmüş olur.) Bir parçacık, koordinatlarını, hızını (çözüm uzayındaki her boyutta ne kadar hızla ilerlediği), şimdiye kadar elde ettiği en iyi uygunluk değerini ve bu değeri elde ettiği koordinatları hatırlamalıdır. Çözüm uzayındaki her boyuttaki hızının ve yönünün her seferinde nasıl değişeceği, komşularının en iyi koordinatları ve kendi kişisel en iyi koordinatlarının birleşimi olacaktır. Bireyler kendi tecrübelerine ve komşularının tecrübelerine göre hareket edecektir. Yani bireyler arasında bilgi paylaşımı sağlanacaktır.

Problemin çözüm uzayı, değişken veya bilinmeyen sayısına bağlı olarak çok boyutta olabilir. Örneğin;

$$5x^2 + 2y^3 - (z/w)^2 + 4 \tag{2.1}$$

fonksiyonunun çözüm uzayı x, y, z ve w bilinmeyenlerinden dolayı 4 boyutludur. Bu problemin çözüm uzayında tanımlanan bir parçacığın pozisyonu 4 koordinat ile P=[x, y, z, w] şeklinde belirtilmektedir.

Problemi ifade eden fonksiyon; PSO algoritması veya parçacıklar için önemsizdir. Örneğin yukarıdaki fonksiyonu sıfıra eşitleyerek bir çözüm bulunabilir.

P=[-1, 0, 3, 1] şeklindeki parçacık için fonksiyon;

$$5x^2 + 2y^3 - (z/w)^2 + 4 = 0 \tag{2.2}$$

olur. Veya P=[3, 3, 8, 1] şeklindeki bir parçacık; x=3, y=3, z=8 and w=1 koordinatları için bir uygunluk fonksiyonunu temsil eder fakat bu değerin 39 olduğu ile ilgilenmez. Görsel olarak insanların resmedemediği 4 veya daha fazla boyutlu karmaşık problemlerde çalışmanın PSO için herhangi bir zorluğu bulunmamaktadır. Örneğin PSO, bir yapay sinir ağının eğitiminde kullanılacaksa ve ağda 50 tane bağlantı ağırlığı ve eşik parametresi mevcutsa, problem 50 boyutlu bir uzayda çözülecektir. PSO'nun 50 boyutta çalışması için hiçbir değişiklik yapmaya gerek yoktur.

PSO, bir grup rasgele çözümle (parçacık sürüsü) başlatılır ve güncellemelerle optimum çözüm bulunmaya çalışılır. Her tekrarlamada (iterasyonda), parçacık konumları, iki en iyi değere göre güncellenir. İlki; o ana kadar parçacığın elde ettiği en iyi çözümü sağlayan koordinatlardır. Bu değer "pbest" olarak adlandırılır ve hafızada saklanmalıdır. Diğer en iyi değer ise, populasyonda o ana kadar tüm parçacıklar tarafından elde edilen en iyi çözümü sağlayan koordinatlardır. Bu değer global en iyidir ve "gbest" ile gösterilir. Örneğin D adet parametreden oluşan n adet parçacık olduğunu varsayalım. Bu durumda populasyon parçacık matrisi eşitlik (2.3)'deki gibidir.

$$x = \begin{bmatrix} x_{11} & x_{12} & \dots & \dots & x_{1D} \\ x_{21} & x_{22} & \dots & \dots & x_{2D} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ x_{n1} & x_{n2} & \dots & \dots & x_{nD} \end{bmatrix}_{n \times D} \tag{2.3}$$

Parçacık sürü matrisindeki i'ninci parçacık aşağıdaki gibi ifade edilir.

$$x_i = [\, x_{i1}, x_{i2}, \dots\dots x_{iD} \,] \tag{2.4}$$

Önceki en iyi uygunluk değerini veren i'ninci parçacığın pozisyonu ($pbest_i$)

$$pbest_i = [p_{i1}, p_{i2}, ... p_{iD}] \tag{2.5}$$

şeklinde ifade edilir. gbest ise her iterasyonda tüm parçacıklar için tektir ve

$$gbest = [p_1, p_2, ... p_D] \tag{2.6}$$

şeklinde gösterilir. i'ninci parçacığın hızı (her boyuttaki konumunun değişim miktarı)

$$v_i = [v_{i1}, v_{i2}, ... v_{iD}] \tag{2.7}$$

olarak ifade edilir. İki en iyi değerin bulunmasından sonra parçacık hızları ve konumları aşağıda verilen (2.8) ve (2.9) nolu denklemlere göre güncellenir.

$$v_i^{k+1} = v_i^k + c_1.rand_1^k.\left(pbest_i^k - x_i^k\right) + c_2.rand_2^k.\left(gbest^k - x_i^k\right) \tag{2.8}$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \tag{2.9}$$

Denklem (2.8)'da, c_1 ve c_2 öğrenme faktörleridir. c_1 ve c_2, her parçacığı pbest ve gbest pozisyonlarına doğru çeken, hızlanma terimlerini ifade eden sabitlerdir. c_1, parçacığın kendi tecrübelerine göre hareket etmesini, c_2 ise sürüdeki diğer parçacıkların tecrübelerine göre hareket etmesini sağlar. Düşük değerler seçilmesi parçacıkların hedef bölgeye doğru çekilmeden önce, bu bölgeden uzak yerlerde dolaşmalarına imkan verir. Ancak hedefe ulaşma süresi uzayabilir. Diğer yandan, yüksek değerler seçilmesi, hedefe ulaşmayı hızlandırırken, beklenmedik hareketlerin oluşmasına ve hedef bölgenin es geçilmesine sebep olabilir. Bu algoritma üzerinde araştırmacıların yaptığı denemelerde $c_1=c_2= 2$ olarak almanın iyi sonuçlar verdiği belirtilmiştir. Denklemdeki $rand_1$ ve $rand_2$, [0,1] arasında düzgün dağılımlı rasgele sayılardır. k iterasyon sayısını, i ise parçacık indisini belirtmektedir.

2.4 Geliştirilmiş PSO ve Çeşitli PSO Versiyonları

PSO'nun şu andaki güncel versiyonu elde edilmeden önce çeşitli denemeler yapılmış ve bir çok yöntem ortaya atılmıştır. Fakat hiçbirisi yukarıda anlatılan güncel PSO algoritması kadar başarılı olamamıştır. Bu yöntemlerin bazıları çok yavaştır, bazıları

ise hızlı fakat yerel optimuma yakınsama ihtimali oldukça fazladır. Daha sonra algoritmanın başarımını arttırmak için Yuhui Shi ve R.C. Eberhart tarafından yapılan çalışmalar sonucunda 1998 yılında Geliştirilmiş PSO algoritması ortaya çıkarılmıştır [2]. Bu algoritmaya göre; parçacık hızlarının güncellenmesini sağlayan (2.8) denklemi aşağıdaki şekle dönüşmektedir.

$$v_i^{k+1} = w.v_i^k + c_1.rand_1^k.\left(pbest_i^k - x_i^k\right) + c_2.rand_2^k.\left(gbest^k - x_i^k\right) \quad (2.10)$$

Burada w eylemsizlik ağırlığıdır. w, 1'e yakın seçilmeli ve her iterasyonda doğrusal olarak azaltılmalıdır. PSO da eylemsizlik ağırlığı global ve yerel arama yeteneğini dengelemek için kullanılır. Büyük eylemsizlik ağırlığı global arama, küçük ağırlık ise yerel arama yapılmasını kolaylaştırır. Eylemsizlik ağırlığının dinamik olarak değiştirilmesiyle arama yeteneği dinamik olarak ayarlanmış olur. Eylemsizlik ağırlığı yerel ve global araştırma arasındaki dengeyi sağlar ve bunun sonucunda yeterli optimal sonuca daha az iterasyonda ulaşılır.

Algoritmanın başarımını arttırmak için araştırmacılar tarafından bir çok yöntem ortaya atılmıştır. Algoritmayla ilgili yapılan bu çalışmaların ortak amacı, parçacıkların yerel çözümlere takılmadan, yeterli optimum çözüme daha hızlı bir şekilde yakınsamasını sağlamaktır. Örneğin ortaya atılan yöntemlerden birinde; pbest ve gbest pozisyonlarının orta noktası alınarak parçacık hızları bu noktaya göre güncellenir. Yani (2.8) denklemindeki 2 toplam terimi tek terime indirgenmiş olur.

Ortaya konan başka bir yöntemde ise parçacıklar, komşularının(gbest) ve kendisinin(pbest) geçmişteki en iyi pozisyonları arasında arama yapması yerine komşularının geçmiş en iyi pozisyonu(gbest) ile en iyi orta pozisyon arasında arama yapar. Yani elde edilen pbest değerlerinin ortalaması alınarak yeni bir değer elde edilir.

$$p_i^* = [p_{i1}^*, p_{i2}^*, ...p_{iD}^*] \quad (2.11)$$

olmak üzere; bireyin en iyi değeri aşağıdaki gibi hesaplanır.

$$p_{ij}^* = (p_{1j} + p_{2j} + ...p_{nj})/n \quad (j=1, 2,....,D) \quad (2.12)$$

Sürüdeki n adet paçacığın her boyuttaki en iyi(pbest) değerlerinin ortalaması alınmakta ve yeni p_i^* vektörü elde edilmektedir. Buna göre (2.10) denklemi aşağıdaki şekle dönüşmektedir.

$$v_i^{k+1} = w.v_i^k + c_1.rand_1^k.\left(p_i^{*k} - x_i^k\right) + c_2.rand_2^k.\left(gbest^k - x_i^k\right) \qquad (2.13)$$

Bu algoritmanın avantajı; her parçacığın sadece sürüdeki en iyi (gbest) parçacığın değil, sürüdeki diğer tüm parçacıkların tecrübelerinden yararlanmasıdır. Parçacıklar, komşularının ve kendisinin geçmişteki en iyi pozisyonları arasında arama yapmaz fakat komşularının geçmiş en iyi pozisyonu (gbest) ile en iyi orta pozisyon p_i^* arasında arama yapar. Parçacıklar diğer parçacıkların tecrübelerinden yararlanarak kendi davranışına karar verir.

PSO algoritması yürütülürken ilerleyen iterasyonlarda en iyi olan parçacığın hızı güncellenirken, (2.10) nolu ifadenin sağ tarafındaki ikinci ve üçüncü terimler sürekli sıfır olur. Bu yüzden başka bir parçacık gbest olana kadar şu andaki en iyi parçacığın konumunda bir değişiklik olmaz (bu sebeble ifadedeki birinci terimde sıfır olur). Diğer parçacıklar da gbest olan parçacığı izleyecekleri için parçacıklar belirli bir yerde kümelenmeye başlar fakat kümelendikleri bu yer her zaman optimum çözüm olmayabilir. Bunu önlemek için ortaya atılan bir yöntemde ise parçacık hızları pbest ve gbest dışında rasgele seçilen bir parçacığın konumuna göre güncellenmektedir. Yani hız güncelleme ifadesine bir terim daha eklenmiştir [6].

$$v_i^{k+1} = w.v_i^k + c_1.rand_1^k.\left(pbest_i^k - x_i^k\right) + c_2.rand_2^k.\left(gbest^k - x_i^k\right) + c_3.rand_3^k.\left(x_R^k - x_i^k\right) \qquad (2.14)$$

Hız güncelleme ifadesine eklenen dördüncü terim sayesinde parçacığın rasgele seçilen bir diğer parçacığı da izlemesi düşünülmüştür. Tabi ki bu izleme oranını c_1, c_2, c_3 sabitleri ve her iterasyonda alınan rasgele sayılar belirleyecektir.

Bir başka versiyonda ise sürü içindeki parçacıklar küçük gruplara ayrılmıştır ve sadece grup içinde bilgi paylaşımı yapılması sağlanmıştır. Örneğin her iterasyonda parçacık, kendisine en yakın olan az sayıdaki parçacık ile bir grup oluşturup, bu grup içerisinde gbest seçilmekte ve parçacık hızları güncellenmektedir.

Araştırmacılar tarafından algoritmanın başarımını artırmak için yapılan çalışmalar günümüzde de devam etmektedir. Çeşitli algoritmaların birleşimiyle elde edilen karma (hibrit) algoritma yapıları ile daha iyi sonuçlar alınabileceği düşünülmektedir. (PSO – Genetik Algoritma), (PSO – Gradient Descent), (PSO – Least Square Error) gibi yapılar hibrit algoritmalara örnek verilebilir. Bu sayede algoritmaların birbirlerine karşı üstün yanları birleştirilerek, çözülecek probleme özgü yeni yaklaşımlar ortaya konmaktadır [5, 7]. Bu tez çalışmasında da karma PSO-GA algoritması üzerinde çalışılmış ve bu algoritma kullanılarak bulanık nöral kontrolör eğitimi gerçekleştirilmiştir. İlerleyen bölümlerde bu algoritmaya ait ayrıntılı bilgi verilmiştir.

2.5 PSO Parametre Kontrolü

PSO da ayarlanması gereken çok fazla sayıda parametre yoktur ancak parametrelerin seçimi oldukça önemlidir. Çözülecek probleme özgü, uygun parametreler seçildiği takdirde algoritmanın başarımı artmaktadır[3, 4]. Aşağıda parametrelerin listesi ve tipik değerleri verilmiştir.

Parçacık sayısı: 20 ile 40 arasındadır. Birçok problem için 10 parçacık kullanmak iyi sonuç elde etmek için yeterlidir. Bazı zor veya özel problemlerde ise 100 veya 200 parçacık kullanılması gerekebilir [12].

Parçacık boyutu: Optimize edilecek probleme göre değişmektedir. (çözüm uzayının boyutu)

Parçacık aralığı(range): Optimize edilecek probleme göre değişmekle birlikte farklı boyutlarda ve aralıklarda parçacıklar tanımlanabilir.

Vmax: Bir iterasyonda, bir parçacık da meydana gelecek maksimum değişikliği(hız) belirler. Genellikle parçacık aralığına göre belirlenir. Örneğin X_1 parçacığı (-10,10) aralığında değişiyor ise Vmax=20 sınırlandırılabilir.

Öğrenme Faktörleri: c1 ve c2 genellikle 2 olarak seçilir. Fakat farklı da seçilebilir. Genellikle c1, c2 ye eşit ve (0-4) aralığında seçilir.

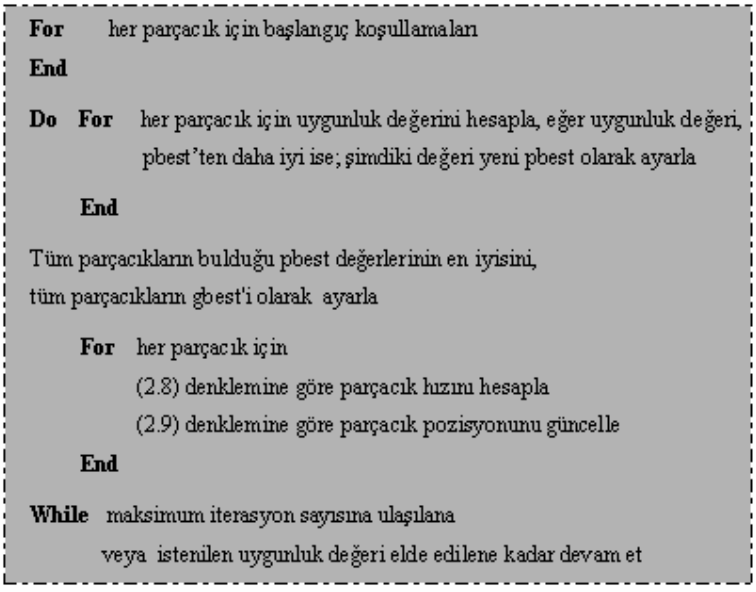
Eylemsizlik Ağırlığı: w; her iterasyonda doğrusal olarak azaltılmalıdır. Genellikle 0.8<w<1.2 seçilebilir.

Durma Koşulu: Maksimum iterasyon sayısına ulaşıldığında veya istenilen uygunluk değerine ulaşıldığında (örneğin minimum hata elde edildiğinde) durdurulabilir. Durdurma koşulu optimize edilecek probleme bağlıdır.

PSO'nun avantajlarından biri, parçacıkların reel değerler almasıdır. GA dan farklı olarak, ikili kodlama gibi değişiklikler ve özel genetik operatörlerin kullanımı yoktur. Örneğin;

$$f(x) = x_1^2 + x_2^2 + x_3^2 \tag{2.15}$$

şeklindeki bir fonksiyon için çözüm aradığımızı varsayalım. Bu durumda parçacıklar (x_1, x_2, x_3) koordinatlarında, uygunluk fonksiyonu ise f(x) olarak ayarlanır. Daha sonra optimumu bulmak için prosedür uygulanır. Arama işlemi tekrarlı bir süreçtir ve durdurma koşulu sağlanana kadar tekrarlanır. Aşağıdaki Şekil 2.1'de PSO algoritması için gerekli olan prosedür görülmektedir.



Şekil 2.1: PSO algoritması için gerekli prosedür

2.6 Benzetim Örnekleri

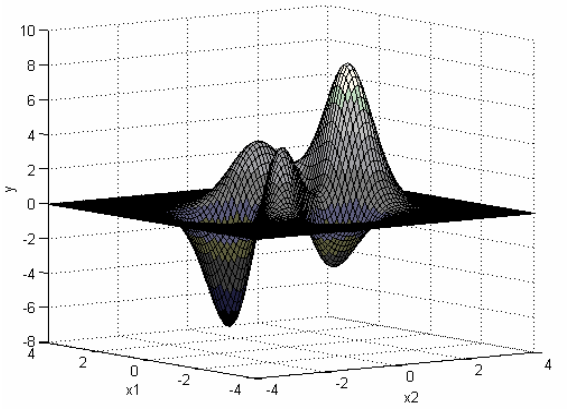
PSO algoritmasının başarımını test edebilmek için; Matlab programı yardımıyla çeşitli benzetimler yapılmıştır. Yapılan benzetimlerde Geliştirilmiş PSO algoritması kullanılmıştır. İlk olarak matematiksel ifadesi bilinen ve birçok yerel maksimum ve minimum içeren "peaks" fonksiyonu tarafından tanımlanan yüzeyde küresel minimum ve maksimum noktası çözümü gerçekleştirilmiştir. İkinci benzetimde ise çok katmanlı bir yapay sinir ağı, EXOR problemini çözmek için PSO ile eğitilmiştir.

2.6.1 Peaks fonksiyonunun global minimum ve maksimum noktalarının bulunması

Bu problemde y fonksiyonu iki değişkene bağlı olduğundan parçacıklar P=[x_1,x_2] şeklinde tanımlanmıştır. Peaks fonksiyonunun matematiksel ifadesi eşitlik (2.16)'da verilmiştir.

$$y = 3(1-x_1)^2.e^{\left[-x_1^2-(x_2+1)^2\right]} - 10.\left(\frac{x_1}{5} - x_1^3 - x_2^5\right).e^{\left[-x_1^2-x_2^2\right]} - \frac{1}{3}.e^{\left[-(x_1+1)^2-x_2^2\right]} \quad (2.16)$$

Benzetimlerde 10 adet parçacık kullanılmış olup PSO'nun minimum noktası için bulduğu çözüm eşitlik (2.17)'de, maksimum noktası için bulduğu çözüm eşitlik (2.18)'de verilmiştir. Bu sonuçların fonksiyonun gerçek min/max noktalarına çok yakın olduğu Şekil 2.2 ve 2.3'ten görülebilir.



Şekil 2.2: Peaks fonksiyonunun tanımladığı yüzey

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0{,}2283 \\ -1{,}6255 \end{pmatrix} \Rightarrow Min_x\, y = -6{,}5511 \tag{2.17}$$

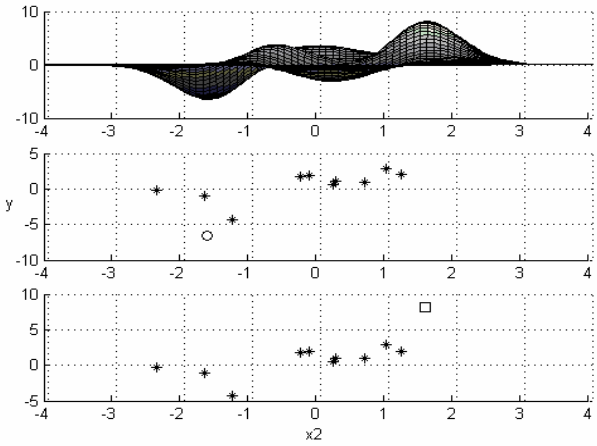
$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} -0{,}0093 \\ 1{,}5814 \end{pmatrix} \Rightarrow Max_x\, y = 8{,}1062 \tag{2.18}$$

Tablo 2.1'de parçacıkların rasgele seçilen başlangıç konumları ve küresel çözümü ararken geldikleri en son konumları görülmektedir. Görüleceği üzere parçacıkların hepsi sürü mantığına uygun olarak global çözümün etrafında toplanmışlardır. Bu durum Şekil 2.3'de de görülmektedir.

Tablo 2.1: Peaks fonksiyonunun çözümü için başlangıç ve sonuç parçacık konumları

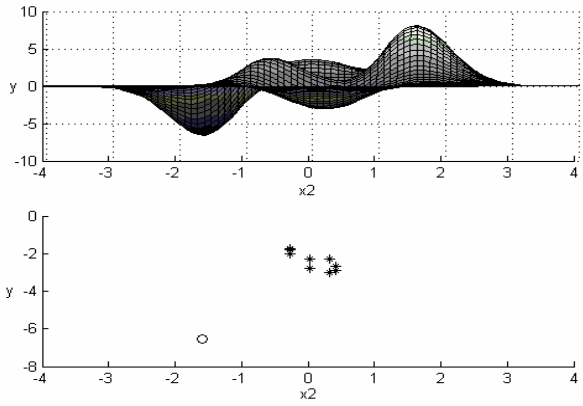
P0 (başlangıç)		**P (min)**		**P (max)**	
X1	**X2**	**X1**	**X2**	**X1**	**X2**
0.0559	0.6804	0.2283	-1.6255	-0.0093	1.5814
-1.1071	-2.3646	0.2283	-1.6255	-0.0093	1.5813
0.4855	0.9901	0.2282	-1.6256	-0.0093	1.5814
-0.0050	0.2189	0.2283	-1.6255	-0.0091	1.5814
-0.2762	0.2617	0.2283	-1.6255	-0.0093	1.5814
1.2765	1.2134	0.2283	-1.6255	-0.0093	1.5814
1.8634	-0.2747	0.2283	-1.6255	-0.0092	1.5814
-0.5226	-0.1331	0.2285	-1.6263	-0.0093	1.5814
0.1034	-1.2705	0.2286	-1.6253	-0.0093	1.5814
-0.8076	-1.6636	0.2282	-1.6256	-0.0094	1.5814

Şekil 2.3'de fonksiyon yüzeyinin ve parçacık pozisyonlarının x düzleminden yan görünüşü verilmiştir. Yıldızla işaretliler parçacıkların başlangıç konumlarını göstermektedir. Yuvarlakla işaretli konum parçacıkların küresel minimum çözümünde son adımda geldikleri yeri (Şekil 2.3b), kare ise parçacıkların küresel maksimum çözümünde son adımda geldikleri yeri göstermektedir (Şekil 2.3c).



Şekil 2.3: (a) Fonksiyonun yüzeyi, (b) Global minimum çözümündeki parçacıkların başlangıç ve son konumları (c) Global maksimum çözümündeki parçacıkların başlangıç ve son konumları

Yapılan benzetimlerde PSO'nun başarımını daha detaylı test etmek için başlangıçta tüm parçacıklar yerel minimuma yerleştirilmiş (Şekil 2.4b yıldızla gösterilenler) ve bu durum için algoritma küresel minimumu bulmak için koşturulmuştur. Sonuç olarak bu başlangıç durumu için bile parçacıklar global minimuma çok yakın bir bölge etrafında toplanmışlardır (Şekil 2.4b yuvarlakla gösterilenler).

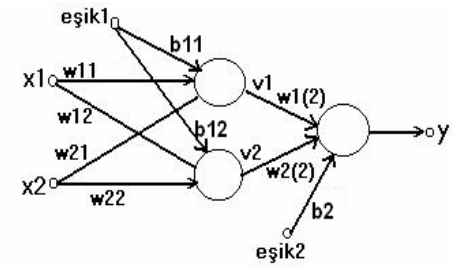


Şekil 2.4: (a) Fonksiyonun yüzeyi, (b) Global minimum çözümündeki parçacıkların başlangıç (yerel minimuma yerleştirilmiş) ve son konumları

Uygulamada en iyi çözüme ulaşmak için tüm parçacıkların bir araya toplanmasını beklemek gerekmez. Her iterasyon sonucunda bulunan gbest değeri istediğimiz değer fonksiyonunu sağlıyorsa, bu gbest değerini bulan parçacık çözüm olarak alınabilir. Bu benzetimleri gerçekleştiren Matlab programları, Ek-1A ve Ek-1B'de verilmiştir.

2.6.2 Çok katmanlı ağ eğitimi

PSO algoritmasının başarımını test etmek için yapılan ikinci benzetimde ise 2 katmanlı bir yapay sinir ağının eğitimi gerçekleştirilmiştir. PSO algoritmasıyla eğitilen ağ ile mantıksal EXOR problemi çözülmeye çalışılmış ve elde edilen sonuç, ağ eğitiminde kullanılan klasik optimizasyon yöntemleriyle karşılaştırılmıştır. EXOR probleminde 2 giriş, 1 çıkış ve 4 adet örnek bulunmaktadır. Bu problem Şekil 2.5'te verilen ağın parametrelerinin PSO ile belirlenmesi sonucunda çözülmüştür. PSO algoritması gradyen bilgisine ihtiyaç duymadığı için hücre aktivasyon fonksiyonu seçiminde esneklik sağlar. Bu uygulamada 0-1 aralığında çıkış veren "sigmoidal" aktivasyon fonksiyonu kullanılmıştır. Sonlandırma kriteri; değer fonksiyonunun binde 1'e inmesi veya maksimum iterasyon sayısına (100) ulaşılması olarak seçilmiştir.



Şekil 2.5: Eğitilecek ağ yapısı

Ölçüt fonksiyonu ise toplam karesel hata olarak eşitlik (2.19)'da verildiği gibi tanımlanmıştır.

$$\cos t = J = \frac{1}{2}\sum_i e_i^2 \tag{2.19}$$

Denklem (2.19)'daki hata (e_i) eşitlik (2.20) ile tanımlıdır.

$$e_i = y_{di} - y_i \tag{2.20}$$

i ise örnek indisini göstermektedir. Uygulamada 60 adet parçacık kullanılmıştır. Ağda ayarlanması gereken eşitlik (2.21)'de verilen 9 adet parametre bulunduğundan, problem 9 boyutlu bir uzayda çözülecektir. Parçacıklar (1x9) yapısında aşağıdaki gibi düzenlenmiştir.

$$P=[w_{11}^1 \;\; w_{12}^1 \;\; w_{21}^1 \;\; w_{22}^1 \;\; b_{11}^1 \;\; b_{12}^1 \;\; w_1^2 \;\; w_2^2 \;\; b_2^2] \tag{2.21}$$

PSO algoritmasıyla ağ eğitiminin benzetimi, Ek-2A'da verilen Matlab programıyla gerçekleştirilmiştir. EXOR problemi için elde edilen sonuçlar Tablo 2.2'de özetlenmiştir. Eğitim sırasında ölçüt fonksiyonunun değişimi Şekil 2.5'te görülmektedir. PSO algoritmasında, kullanılan parçacık sayısına da bağlı olarak 100 iterasyondan kısa süreli eğitimlerle iyi sonuçlar elde edilmektedir. Çok sayıda parçacık kullanmak yakınsamayı hızlandırmakta, ancak işlem yükünü arttırmaktadır.

Klasik optimizasyon yöntemleriyle karşılaştırma yapabilmek için Ek-2B'de verilen program kullanılarak benzetimler gerçekleştirilmiştir. Geriye yayılım algoritması ile yapılan eğitimde öğrenme adımı 0.5 olarak alınmıştır. Daha büyük seçilen öğrenme oranı yakınsamayı hızlandırabileceği gibi, çözümün es geçilmesine ve salınımlara da neden olabilmektedir. Geriye yayılım algoritmasının başarımı başlangıç koşullarına oldukça bağlıdır. Eğer iyi bir yerden aramaya başlanırsa kısa sürede çözüme ulaşılabilmektedir. Ancak bu süre (iterasyon) 1000'ler mertebesinde olmaktadır. Başlangıç koşullarına bağlı olarak çözüme hiç ulaşılamadığı da olabilir. Momentumlu geriye yayılım algoritmasıyla da benzer sürelerde sonuç alınmaktadır.

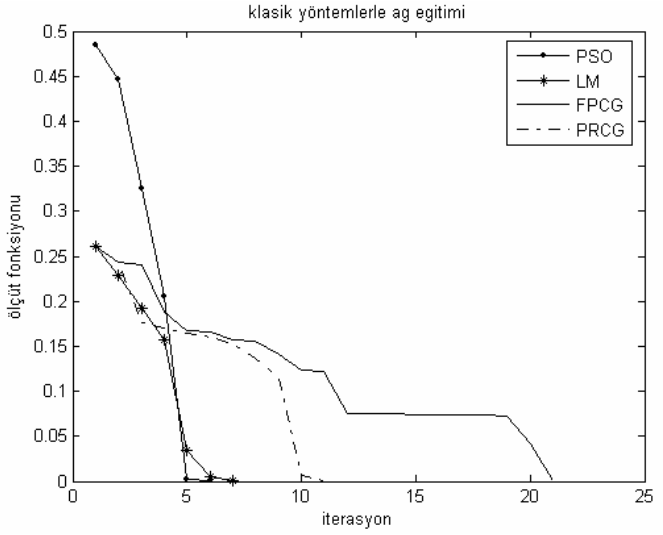
Levenberg-Marquardt (LM), Fletcher-Powell Conjugate Gradient (FPCG), Polak-Ribiére Conjugate Gradient (PRCG) gibi klasik yöntemlerle başlangıç konumuna bağlı olarak oldukça hızlı sonuçlar elde etmek mümkündür. Örneğin aşağıda verilen parametrelerden başlanarak, yukarıda verilen yöntemlerle, Matlab Neural-Network Toolbox yardımıyla, Ek-2B'de verilen program kullanılarak gerçekleştirilen benzetimlerde elde edilen sonuçlar Tablo 2.2'de ve Şekil 2.6'da verilmiştir. Geriye yayılım algoritması kullanıldığında eğitim süresi diğer yöntemlere göre çok uzun olduğu için Şekil 2.6'da bu benzetime ait ölçüt fonksiyonunun değişimine yer verilmemiştir.

Başlangıç parametreleri:

- Giriş-ara katman arası bağlantı ağırlıkları:
  $W1=[w_{11}\ w_{12};\ w_{21}\ w_{22}]=[-1.5804\quad -0.6817;\ -0.0787\quad -1.0246]$
- Ara katman-çıkış arası bağlantı ağırlıkları: $W2=[w_1\ w_2]=[-1.2344\quad 0.2888]$
- Ara katman hücre eşikleri: $B1=[b_{11};\ b_{12}]=[-0.4293;\ 0.0558]$
- Çıkış katmanı hücre eşiği:$B2=[b_2]=[-0.3679]$

Tablo 2.2: EXOR problemi için elde edilen sonuçlar

X1	X2	Yd	PSO	GY	LM	FPCG	PRCG
0	0	0	0.0000	0.0353	0.0324	0.0002	0.0047
0	1	1	0.9986	0.9614	0.9925	0.9747	0.9775
1	0	1	0.9963	0.9613	0.9695	0.9710	0.9994
1	1	0	0.0003	0.0487	0.0402	0.0448	0.0376
iterasyon			6	5000	6	20	10
Ölçüt fonksiyonu			0,0000078	0.00165	0.00091	0.00087	0.00048



Şekil 2.6: Ağ eğitimleri sırasında ölçüt fonksiyonunun iterasyonla değişimi

Klasik yöntemlerle bu başlangıç konumu için oldukça kısa iterasyon sayılarında iyi sonuçlar elde edildiği görülmüştür. Ancak her zaman bu kadar kısa sürede sonuç alınamamaktadır. Uzun süren eğitimler sonucunda çözüme erişilemeye de bilir. PSO algoritmasının avantajı, başlangıçta bir çok aday çözüm noktasından aramaya başlamasıdır.

Bir problem için, algoritmanın vereceği sonuç, problemin karmaşıklığı, parametre sayısı, eldeki veri sayısı, kullanılan ağ yapısı, başlangıç konumu gibi bir çok nedene bağlıdır. Bu nedenle algoritmaların birbirlerine karşı üstünlükleri hakkında kesin bir şey söylemek zordur. Bu konuda; farklı problemler üzerinde, farklı başlangıç noktalarından başlanarak çok sayıda benzetim sonuçları içeren daha geniş kapsamlı çalışmalar yapılması gerekir.

Burada tanıtılan PSO algoritması klasik yöntemlere alternatif olarak ortaya konmuş bir yöntemdir. Gerçekleştirilen benzetimlerde algoritmanın çok boyutlu problemlere oldukça hızlı ve yeterli sonuçlar verdiği görülmüştür. İşlem yükü ve gerekli hafıza miktarı kullanılan parçacık sayısına ve probleminin boyutuna bağlıdır. Ancak algoritma; herhangi bir diferansiyel denklem çözümü ve gradyen hesabı gerektirmediğinden, ayrıca ayarlanması gereken az sayıda parametre bulunduğundan yürütülmesi kolaydır. PSO'nun; fonksiyon optimizasyonunda, global çözüme kolayca ulaştığı yapılan benzetimlerde görülmüştür. Yapılan benzetimlerde EXOR problemini çözebilmek için gerekli ağ parametreleri PSO algoritması ile oldukça hızlı bir şekilde belirlenmiştir. Bu sonuçla, yapay sinir ağlarının eğitimi için kullanılan gradyen tabanlı klasik öğrenme algoritmaları yerine PSO'nun kullanılabileceği görülmüştür. Hibrit algoritma yapılarıyla, PSO algoritmasının başarımının artırılabileceği düşünülmektedir. İlerleyen bölümlerde bu doğrultuda yapılan çalışmalar anlatılmaktadır.

BÖLÜM 3: KARMA RASGELE ARAMA ALGORİTMASI

3.1 Giriş

PSO algoritmasının başarımını arttırmak için araştırmacılar tarafından bir çok yöntem ortaya atılmıştır. Algoritmayla ilgili yapılan bu çalışmaların ortak amacı, parçacıkların yerel çözümlere takılmadan, yeterli optimum çözüme daha hızlı bir şekilde yakınsamasını sağlamaktır. Algoritma yürütülürken ilerleyen iterasyonlarda en iyi olan parçacık hızı güncellenirken, (2.8) nolu ifadenin sağ tarafındaki ikinci ve üçüncü terimler sürekli sıfır olur. Bu yüzden başka bir parçacık gbest olana kadar şu andaki gbest olan parçacığın konumunda bir değişiklik olmaz (bu sebeble ifadedeki birinci terimde sıfır olur). Diğer parçacıklar da gbest olan parçacığı izleyecekleri için parçacıklar belirli bir yerde kümelenmeye başlar fakat kümelendikleri bu yer her zaman optimum çözüm olmayabilir. Bunu önlemek için araştırmacılar tarafından çeşitli çalışmalar yapılmış ve günümüzde de yapılmaya devam etmektedir. Çeşitli algoritmaların birleşimiyle elde edilen hibrit algoritma yapıları ile daha iyi sonuçlar alınabileceği düşünülmektedir. Örneğin Gradient Descent (GD) metoduyla PSO'nun birleştirilmesiyle hibrit bir algoritma ortaya konmuştur [7]. PSO algoritmasına benzer şekilde populasyon tabanlı bir algoritma olan Genetik Algoritmanın(GA) PSO ile birleştirilmesiyle de hibrit yapılar elde edilmiştir [5]. Bu sayede algoritmaların birbirlerine karşı üstün yanları birleştirilerek, çözülecek probleme özgü yeni yaklaşımlar ortaya konmaktadır. Bu bölümde PSO ile GA'nın birleşiminden elde edilen karma (hibrit) algoritma anlatılacaktır. Bu Karma arama algoritması, "Hibrit Parçacık Sürüsü Genetik Algoritma" (HPSOGA) olarak adlandırılmaktadır. İlerleyen bölümlerde verilecek olan benzetim sonuçlarından; bu yeni algoritmanın klasik PSO algoritmasından daha iyi sonuçlar verdiği görülmüştür.

3.2 Genetik Algoritma

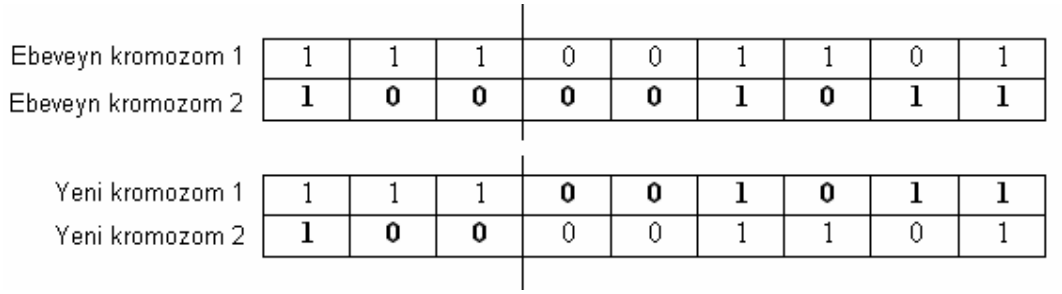
1975 yılında John Holland'ın "Doğal ve yapay sistemlerin uyumu" isimli kitabıyla ortaya çıkan Genetik Algoritma; rasgele aramaya dayalı bir yöntemdir. GA, doğanın biyolojik evrim modelini baz alarak çözüm arayan bir arama algoritmasıdır. Biyolojik evrime benzetildiğinden üreme, çaprazlama, mutasyon gibi operatörler içermektedir. Genetik algoritmaların, fonksiyon optimizasyonu, çizelgeleme, mekanik öğrenme, haberleşme şebekeleri tasarımı, gaz boruları şebekeleri optimizasyonu, görüntü ve ses tanıma, veri tabanı sorgulama optimizasyonu, fiziksel sistemlerin kontrolü, gezgin satıcı problemlerinin çözümü, ulaşım problemleri, optimal kontrol problemleri gibi birçok kullanım alanı vardır [16, 17].

Genetik algoritmalar da PSO algoritması gibi, geleneksel olmayan optimizasyon teknikleri arasında yer almaktadır. PSO algoritmasıyla benzer şekilde; genetik algoritmalar, çözüm bilgisinin hiç olmadığı veya çok az olduğu bir durumla arama yapmaya başlar. Çözüm çevreden gelen etkileşime ve genetik operatörlere bağlıdır. Yapılan arama işlemi paralel biçimdedir ve birbirinden bağımsız noktalardan başlamaktadır. Bu nedenle algoritmanın, yerel çözüm noktalarına takılma olasılığı azdır.

GA'da olası çözümler kromozomlarla ifade edilir. Her kromozom belirli sayıda gen içerir. PSO ile benzerlik kurulması açısından kromozomları parçacıklara, genleri parçacığı oluşturan parametrelere benzetebiliriz. Hibrit algoritma; PSO algoritması içersine GA'daki çaprazlama (crossover) ve mutasyon (değişim) operatörlerinin eklenmesiyle oluşturulmuştur. Çaprazlama ve değişim operatörleri aşağıda açıklanmaya çalışılmıştır.

Çaprazlama Operatörü: Bir önceki nesilde iyi uygunluk değerine sahip olan bireyler arasından çeşitli yöntemlerle seçilen 2 adet kromozomun (parent/ebeveyn) çaprazlanmasıyla 2 adet yeni kromozom elde edilir ve bir sonraki nesil için, kötü uygunluk değerine sahip bireylerin yerine sürüye dahil edilir.

Aşağıdaki Şekil 3.1'de 2'li olarak kodlanmış kromozomlar için tek noktalı basit bir çaprazlama işlemi görülmektedir.



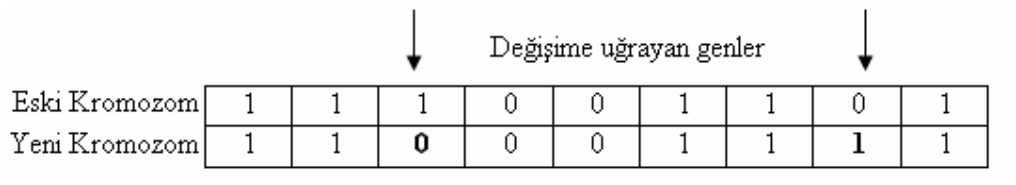
Şekil 3.1: Tek noktalı çaprazlama

Çaprazlama işlemi; Tek nokta çaprazlama, İki nokta çaprazlama, Çok nokta çaprazlama, Uniform çaprazlama gibi farklı şekillerde yapılabilir. Çaprazlama işlemi için 2 adet parent (ebeveyn) bireyin seçimi de farklı şekillerde yapılabilir. En çok kullanılan yöntemler: Rulet tekerleği yöntemi, Turnuva seçimi ve Elitist seçim yöntemleridir.

Turnuva seçimi yönteminde her bireyin yarışacağı belirli sayıda rasgele seçilmiş bireylerden bir alt popülasyon oluşturulur. Birey, alt popülasyondaki bireylerle uygunluk değerlerine göre yarışır ve kim kazanırsa çaprazlama için seçilir.
Elitist seçim yönteminde popülasyonun en iyi bireyi korunur. Popülasyonun geri kalan elemanları, seçim yöntemlerinden birisi ve genetik operatörler kullanılarak yeni bireyler ile değiştirilir. Burada hedef en iyi uygunluk değerine sahip bireyin, genetik operatörler kullanıldığında kaybolmasını önlemektir.

Çaprazlamanın amacı, mevcut iyi kromozomların özelliklerini birleştirerek daha uygun kromozomlar yaratmaktır. Kromozom çiftleri çaprazlama olasılığı (Pc) ile çaprazlamaya uğramak üzere seçilirler. Çaprazlamanın artması, yapı bloklarının artmasına neden olmakta fakat aynı zamanda bazı iyi kromozomların da bozulma olasılığını arttırmaktadır. Düşük çaprazlama oranı yeni kuşağın üretilmesi için çok az sayıda bireyin çaprazlanmasına sebep olmaktadır. Yüksek çaprazlama oranı araştırma uzayının çok hızlı bir şekilde araştırılmasına sebep olmaktadır. İleride anlatılacak olan hibrit algoritma içerisinde; turnuva seçimi yöntemi ve iki noktalı çaprazlama kullanılmıştır.

Değişim Operatörü: Çaprazlama operatörüyle elde edilen yeni popülasyondaki kromozomlara ait genler tek tek kontrol edilir ve mutasyon oranına (Pm) göre değiştirilir. İkili kodlanmış bir genin değeri 1 ise 0, 0 ise 1 olarak değiştirilir. Bu operatörün nasıl çalıştığı Şekil 3.2'de gösterilmektedir.



Şekil 3.2: Mutasyon (Değişim) operatörünün işleyişi

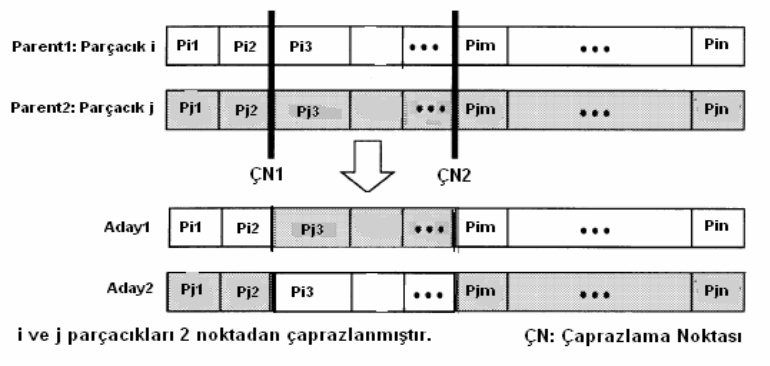
Mutasyonun amacı populasyondaki genetik çeşitliliği korumaktır. Mutasyon, mutasyon olasılığı ile bir kromozomdaki her bitte meydana gelebilir. Eğer mutasyon olasılığı artarsa, genetik arama rastsal bir aramaya dönüşür. Fakat bu aynı zamanda kayıp genetik malzemeyi tekrar bulmada yardımcı olmaktadır.Yüksek mutasyon oranı, araştırmaya aşırı bir rasgelelik kazandıracak ve araştırmayı çok hızlı olarak ıraksatacaktır. Başka bir deyişle, populasyonun gelişmesine değil tahribatına sebep olacaktır. Bu durumun tersine, çok düşük mutasyon oranının kullanılması ıraksamayı aşırı düşürecek ve araştırma uzayının tamamen araştırılmasını engelleyecektir. Mutasyon operatörü, araştırma sahasına yeni bölgelerin girmesini sağlayarak; yeni, görülmemiş ve araştırılmamış çözüm elemanlarının bulunmasına sebep olur. Böylece; algoritmanın alt optimal çözümlere takılması engellenmiş olur. Mutasyon işlemi: Ters çevirme, Ekleme, Yer değişikliği, Karşılıklı değişim gibi çeşitli şekillerde gerçekleştirilebilir.

Bu çalışma içersinde GA ile ilgili fazla ayrıntılı bilgiye yer verilmemiştir, sadece hibrit algoritma içersinde GA'nın rolüne değinilmeye çalışılmıştır. GA ile ilgili daha ayrıntılı bilgi için [16, 17]'dan yararlanılabilinir.

3.3 Karma Arama Algoritması

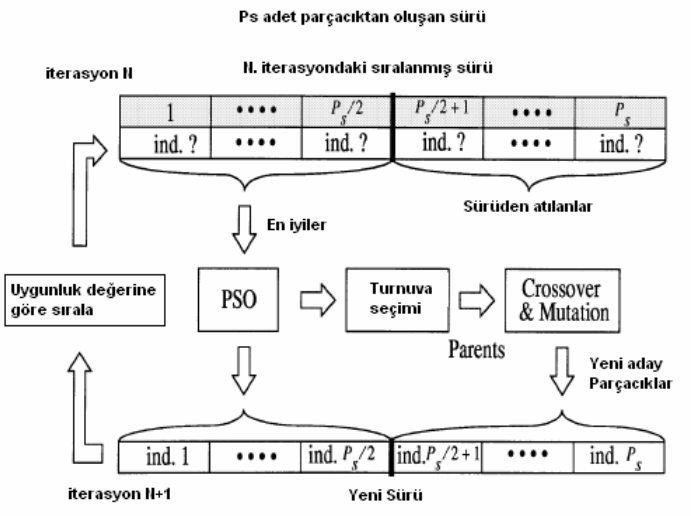
Günümüzde giderek daha karmaşık hale gelen koşullar, hızlı ve kolay çözüm veren yeni çözüm yöntemleri arayışına neden olmuştur. Özellikle hard optimization (sert optimizasyon) teknikleri yerine soft computing (yaklaşık hesaplama) ve evrimsel algoritma (evolutionary algorithm) kullanımı ön plana çıkmıştır. Evrimsel yaklaşımlardan olan Genetik Algoritmalar da bu arayışlarda önemli bir role sahip hale gelmiştir. Uygulama başarısı yüksek olan ve sürekli geliştirilmeye çalışılan genetik algoritmalar, diğer yaklaşık hesaplama yöntemleri kullanılarak hibrit (karma) çözümler geliştirilmeye çalışılmaktadır. Karma rasgele arama algoritmaları, klasik yöntemlerin çok uzun zamanda yapacakları işlemleri kısa bir zamanda çok net olmasa da yeterli bir doğrulukla yapabildikleri için optimizasyon alanında sıklıkla kullanılmaktadır.

Yukarıda da bahsedildiği gibi hibrit algoritma da, sürüdeki bireyler güncellenirken PSO algoritmasına, GA'daki çaprazlama ve değişim operatörleri dahil edilmiştir. Öncelikle, klasik PSO algoritmasında olduğu gibi, sürüdeki tüm parçacıklar sisteme uygulanır ve parçacıkların uygunluk değerleri hesaplanır. Hesaplanan bu uygunluk değerine bağlı olarak parçacıklar için pbest ve gbest değerleri belirlenir. Parçacıklar uygunluk değerlerine göre iyiden kötüye doğru sıralanır. Sürüdeki parçacıkların iyi uygunluk değerine sahip olan yarısı seçilir, kötü uygunluk değerine sahip olan parçacıklar ise sürüden atılır. İyi uygunluk değerine sahip olan parçacıklar arasından, GA'daki "turnuva seçimi" yöntemiyle, çaprazlama olasılığına bağlı olarak 2 adet ebeveyn parçacık seçilir. Bu işlem şu şekilde gerçekleştirilir. İyi uygunluk değerine sahip bireyler için rasgele sayılar üretilir ve çaprazlama oranından büyük sayıya sahip olan bireyler bir alt grup oluşturur. Bu grup içerisinde turnuva seçimi yöntemine göre daha iyi uygunluk değerine sahip olan birey ebeveyn olarak seçilir. Aynı işlemin tekrarlanması ile 2. ebeveyn de seçilir. Seçilen 2 adet ebeveyn parçacığın 2 noktalı çaprazlanması ile 2 adet yeni parçacık elde edilir. Çaprazlama işlemi Şekil 3.3'de gösterilmiştir.



Şekil 3.3: İki noktalı çaprazlama işlemi

Çaprazlama işlemiyle elde edilen yeni parçacıkların bazı parametreleri değişim olasılığına bağlı olarak değiştirilir. Burada parçacıkları oluşturan parametreler reel sayılardan oluşmaktadır. Yani herhangi bir 2'li kodlama söz konusu değildir. Bu sebeple mutasyon işlemi yukarıda anlatılandan biraz farklı ele alınmıştır. Her yeni parçacık için, parçacığın boyutunda (parametre sayısı kadar) rasgele sayı üretilir. Raslantısal olarak mutasyon olasılığından küçük sayıya sahip olan parametreler rasgele olarak değiştirilir. Eğer problem uzayında ilgili parametre ile ilgili bir kısıtlama varsa, bu parametreye atanan bu rasgele yeni değer de bu kısıtlamaları sağlamalıdır. Elde edilen bu yeni bireyler bir sonraki iterasyon için sürüye dahil edilir. Sürüdeki iyi uygunluk değerine sahip olan bireyler ise PSO algoritmasıyla (2.8) ve (2.9) eşitliklerine göre güncellenir. Hibrit algoritmanın işleyişi Şekil 3.4'de görülmektedir.



Şekil 3.4: HPSOGA algoritmasının işleyişi [5]

Algoritmanın her iterasyonunda gbest'ten bağımsız yeni aday parçacıkların sürüye dahil edilmesiyle, parçacıkların yerel çözümlere takılması engellenmiş olup, yeterli optimum çözümün daha çabuk bulunması sağlanmıştır. Hibrit algoritmanın işleyişi için gerekli prosedür Şekil 3.5'te verilmiştir.

```
For     her parçacık için başlangıç koşullamaları
End
Do  For     her parçacık için uygunluk değerini hesapla, eğer uygunluk değeri,
            pbest'ten daha iyi ise; şimdiki değeri yeni pbest olarak ayarla
    End
    Tüm parçacıkların bulduğu pbest değerlerinin en iyisini,
    tüm parçacıkların gbest'i olarak ayarla,
    Parçacıkları uygunluk değerlerine göre sırala,
    Sürüdeki parçacılarından, kötü uygunluk değerine sahip olan yarısını at
    For     sürüdeki birey sayısının dörtte biri kadar
            Sürüdeki iyi uygunluk değerine sahip olan bireyler arasından 2 adet
            ebeveyn seç, çaprazlama ve değişim operatörleri ile 2 adet yeni birey
            elde et. (Genetik Algoritma)
    End
    For     sürüdeki birey sayısının yarısı kadar
            Sürüye dahil edilen yeni bireylerin başlangıç koşullamaları
    End
    For     sürüdeki birey sayısının yarısı kadar
            Sürüdeki iyi uygunluk değerine sahip olan parçacıkların hız ve
            konumlarını PSO algoritmasına göre güncelle
    End
While   maksimum iterasyon sayısına ulaşılana
    veya istenilen uygunluk değeri elde edilene kadar devam et
```

Şekil 3.5: Hibrit algoritma için gerekli prosedür

BÖLÜM 4: BULANIK – NÖRAL AĞLAR (FUZZY NEURAL NETWORKS)

4.1 Giriş

Bu bölümde çeşitli bulanık-nöral ağ yapıları anlatılacaktır. Öncelikle adaptif bulanık çıkarım sistemleri için temel olan ANFIS ağ yapısı tanıtılacaktır. Daha sonra ise 2 adet yinelemeli ağ yapısı üzerinde durulacaktır.

- ANFIS (Adaptive – Network – Based Fuzzy Inference System)
- RFNN1: TRFN (Takagi-Sugeno-Kang(TSK) Recurrent Fuzzy Network)
- RFNN2 (Recurrent Fuzzy Neural Network)

Ağın katmanları ve ağda ayarlanması gerekli olan parametreler açıklanacaktır. Bu ağ yapıları ilerleyen bölümlerde de açıklanacağı üzere Parçacık Sürüsü algoritmasıyla, kontrolör olarak eğitilecektir.

4.2 ANFIS : Uyarlamalı Ağ Tabanlı Bulanık Çıkarım Sistemi

Bu bölümde Sugeno bulanık modeline eşdeğer ANFIS yapısı anlatılacaktır. Bu türden ağ yapıları "Uyarlamalı Yapay Sinir Ağı Tabanlı Bulanık Çıkarım Sistemi" veya anlamca eşiti olan "Adaptif Bulanık-Nöral Çıkarım Sistemi" anlamına gelen "ANFIS" olarak bilinmektedir [18]. Bu tür ağ yapıları bulanık çıkarım sistemleri için temel oluşturmaktadır.

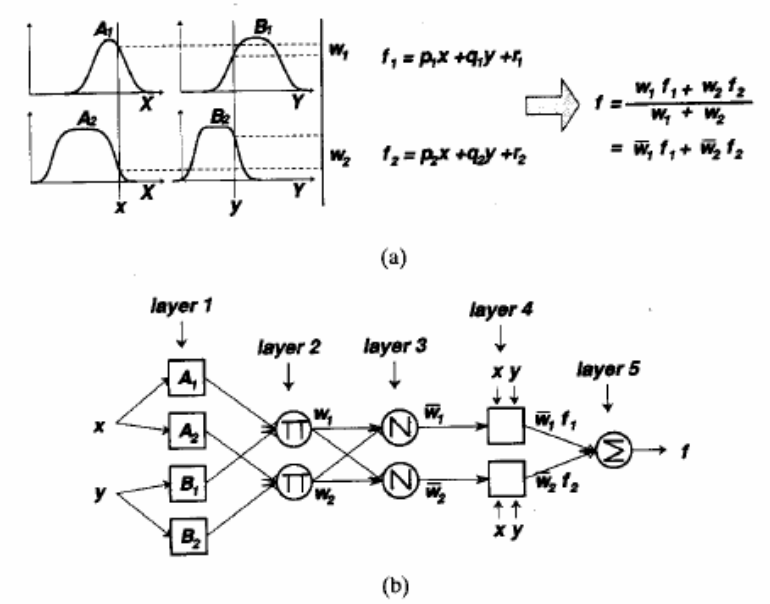
Burada ANFIS yapısındaki bulanık çıkarım sisteminin kolaylıkla anlaşılabilmesi için, x ve y gibi iki girişi ve z gibi bir tane çıkışı olan birinci dereceden Sugeno bulanık modeli ele alınmıştır.

Birinci dereceden Sugeno bulanık modeli için, kural kümesindeki tipik iki adet "IF - THEN (Eğer – O Halde)" kuralı şöyle ifade edilebilir :

Kural 1 : Eğer $x=A_1$ ve $y=B_1$ ise O halde $f_1=p_1x + q_1y + r_1$ (4.1)

Kural 2 : Eğer $x=A_2$ ve $y=B_2$ ise O halde $f_2=p_2x + q_2y + r_2$ (4.2)

Şekil 4.1'de; bu Sugeno Bulanık modeli için çıkarım mekanizması görülmektedir (a). Ve buna karşılık gelen ANFIS ağ yapısı görülmektedir (b).



Şekil 4.1: İki kurallı, iki girişli, birinci dereceden Sugeno bulanık modeli, (b); Buna karşılık gelen ANFIS mimarisi [17]

Aşağıda, ağın her bir katmandaki düğümlerin çıkışları verilmiştir. Burada l tabakasındaki i çıkış düğümü $O_{l,i}$ olarak gösterilmiştir.

1. Katman: Bu katmandaki her bir düğüm A_i ve B_i gibi bir bulanık kümeyi ifade eder. Bu katmandaki düğümlerin çıkışı giriş örneklerine ve kullanılan üyelik fonksiyonlarına bağlı olan üyelik dereceleridir. Bu katmandaki her bir i düğümünün çıkışı aşağıdaki gibi tanımlanmıştır.

$O_{1,i} = \mu_{Ai}(x)$, i = 1,2 için veya (4.3)

$O_{1,i} = \mu_{B\,i-2}(y)$, i = 3,4 için (4.4)

Burada, A_i ve B_i bulanık kümeleri için üyelik fonksiyonu herhangi bir uygun parametreli üyelik fonksiyonu olabilir. Örneğin, A_i bulanık kümesi Gauss biçimli üyelik fonksiyonu ile tanımlanabilir.

$$\mu_{Ai}(x) = \exp\left[-\left(\frac{x-c_i}{\sigma_i}\right)^2\right] \tag{4.5}$$

Bu denklemde $\{\sigma_i, c_i\}$ Gauss fonksiyonunun varyansını ve merkezini belirleyen parametrelerdir. Bu tabakadaki parametreler şart parametreleri olarak adlandırılır. Bu parametreler adaptif ağın eğitimi sırasında en uygun şekilde ayarlanabilir.

2. Katman: Bu tabakadaki her bir düğüm (Π) ile tanımlanmış olup, gelen işaretleri çarparak sonuca ulaşan bir düğümdür. Bu katmandaki her bir düğümün çıkışı, bir kuralın aktiflik derecesini (ateşleme gücü) vermektedir. Aslında bu katmanda T-norm işlemi gerçekleştirilmektedir ve herhangi bir T-norm operatörü bu katmanda düğüm fonksiyonu olarak kullanılabilir.

$$O_{2,i} = w_i = \mu_{Ai}(x) * \mu_{Bi}(y),\ i = 1,2 \tag{4.6}$$

3. Katman: Bu katmandaki her bir düğüm N ile tanımlanmaktadır. i. düğüm , bütün kuralların ateşleme güçleri toplamının, i. kuralın ateşleme gücüne oranını hesaplar. Bu tabakadaki çıkışlar normalleştirilmiş ateşleme güçleri olarak adlandırılır.

$$O_{3,i} = \overline{w_i} = \frac{w_i}{w_1 + w_2}, i = 1,2 \tag{4.7}$$

4. Katman: Bu katmandaki her bir düğüm, düğüm işlevi ile adaftiftir. Buradaki her bir i düğümü ilgili kuralın sonuç üzerindeki etkisini hesaplar.

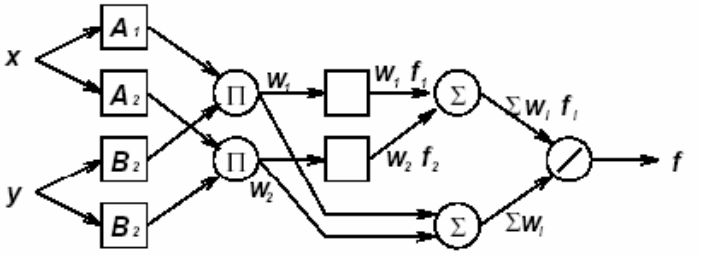
$$O_{4,i} = \overline{w_i}.f_i = \overline{w_i}.(p_i x + q_i y + r_i) \tag{4.8}$$

Burada $\overline{w_i}$, 3. katmanın çıkışıdır. { p_i , q_i , r_i } ise parametre kümesidir. Bu katmandaki parametreler sonuç parametreleri olarak adlandırılır.

5. Katman**:** Bu katmandaki tek düğüm Σ ile tanımlanmış olup, gelen bütün işaretleri toplayarak ağ çıkışını hesaplar.

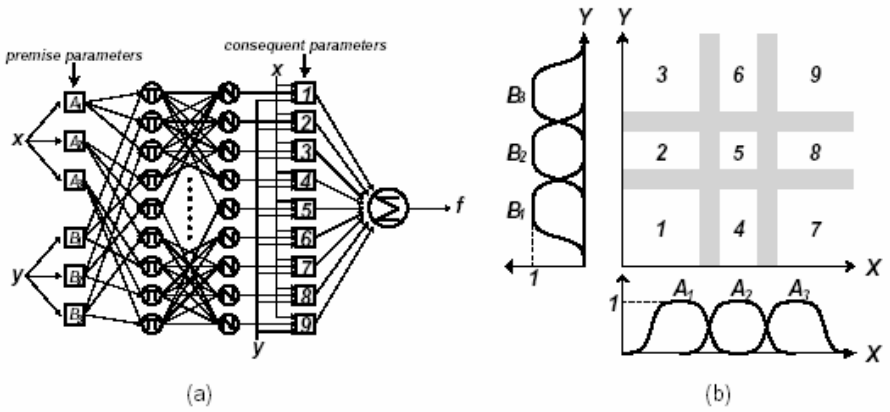
$$O_{5,i} = z = \sum_i \overline{w_i} . f_i = \frac{\sum_i w_i . f_i}{\sum_i w_i} \tag{4.9}$$

Yukarıda Sugeno bulanık çıkarım modeline işlevsel olarak eşdeğer olan ANFIS yapısı açıklanmaya çalışılmıştır. Bu adaptif ağ yapısı tek değildir. 3.katman ile 4. katman sadece 4. katmanda birleştirilmesi ve son katmanda bir ağırlık normalizasyonu yapılmasıyla işlevsel olarak denk bir ağ yapısı elde edilebilir. Aşağıdaki şekilde bu türden bir ANFIS yapısı görülmektedir.



Şekil 4.2: İki girişli iki kurallı bir Sugeno bulanık modeli için bir başka ANFIS mimarisi

Şekil 4.3'te dokuz kurallı ve her bir girişinin, üç üyelik fonksiyonuna sahip olduğunun varsayıldığı iki girişli, birinci dereceden Sugeno bulanık modeline eşit olan bir ANFIS yapısı görülmektedir (a). (b)' de ise, iki boyutlu giriş uzayının , her bir bağıntının bulanık "if-then" kuralları tarafından dokuz tane üst üste binmiş bulanık bölgeye ayrılmasını açıklanmaktadır. Başka bir deyişle, bir kuralın şart kısmı, kuralın sonuç kısmı bu bölgedeki bir çıkışı belirttiği sürece, bir bulanık bölgeyi tanımlamaktadır.



Şekil 4.3: İki girişli dokuz kurallı birinci dereceden bir Sugeno Fuzzy modeli için ANFIS mimarisi [18]

Mamdani ve Tsukamoto bulanık modelleri için de benzer ANFIS yapıları oluşturulabilir. Ancak bu çalışma içersinde bunlara değinilmemiştir. ANFIS yapısı hakkında daha ayrıntılı bilgi almak için [18]'den yararlanılabilinir.

4.3 RFNN1 (TRFN) Yapısındaki Ağ (Yinelemeli Bulanık – Nöral Ağ)

Zamansal bilgi işleme problemine; kontrol, haberleşme, örüntü tanıma gibi bir çok alanda rastlanmaktadır. Bu tarz problemleri çözmenin etkili bir yolu yinelemeli ağ yapıları kullanmaktır [5]. Bu tür ağlarda geçmiş zamana ait bilgilerin kullanılması için geri besleme döngüleri bulunmaktadır. Tez içersinde bu ağdan TRFN olarak bahsedilecektir. Şekil 4.4'te TRFN ağ yapısı görülmektedir. Ağın her katmanındaki düğümlerin çıkışları ve eğitim sırasında ayarlanması gereken parametreler aşağıda sırasıyla açıklanmıştır.

1. Katman: Ağın birinci katmanında harici giriş değişkenleri ve geçmişteki kural çıkışlarına ait geri besleme değişkenleri (dahili) bulunmaktadır. Bu katmanda herhangi bir parametre bulunmamaktadır.

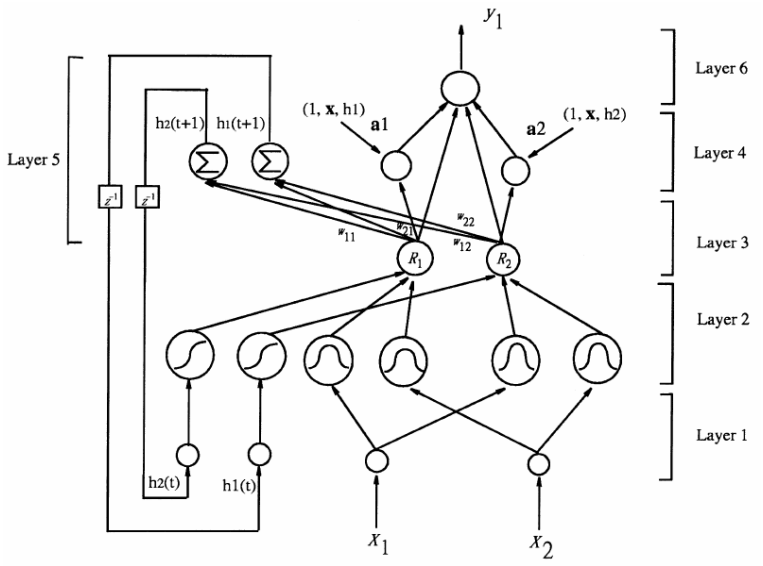
2. Katman: İkinci katmandaki her bir düğüm bir üyelik fonksiyonunu temsil eder. Örneğin harici girişler için Gauss biçimli (4.5), dahili değişkenler için sigmoidal üyelik fonksiyonları (4.10) kullanılabilir.

$$S(x) = \frac{1}{1 + e^{-s(x-c)}} \tag{4.10}$$

Bu katmanda eğitim sırasında ayarlanması gereken; üyelik fonksiyonlarına ait merkez, varyans, eğim gibi parametreler bulunmaktadır.

3. Katman: Bu katmanda kural düğümleri bulunmaktadır. Her kural düğümü harici girişlerin ÜF'lerine ve o kuralın daha önceki çıkışına bağlı bir çıkış verir. Yani bu katmanda T-norm işlevi gerçekleştirilerek kural aktiflik derecesi hesaplanmış olur.

$$\begin{aligned} R_i(t) &= \mu(x(t), h_i(t)) \ldots\ldots \\ \ldots\ldots &= \frac{1}{1 + e^{-s(h_i(t)-c)}} . \exp\left[- \sum_{j=1}^{n} \left(\frac{x_j(t) - c_{ij}}{\sigma_{ij}} \right)^2 \right] \end{aligned} \tag{4.11}$$



Şekil 4.4: TRFN yapısındaki ağ [5]

4. Katman: Dördüncü katmandaki düğümlerde, her kural için, harici giriş değişkenlerinin, geri besleme değişkeninin ve bir sabitin ağırlıklı doğrusal birleşiminden bir çıkış elde edilir (4.12). Burada ayarlanması gereken, her bir kural için, n giriş değişkeni için (n+2) adet (a_{ij}) parametre bulunmaktadır.

$$f_i(t) = a_{i0} + \left(\sum_{j=1}^{n} a_{ij}.x_j(t) \right) + a_{i(n+1)}.h_i \tag{4.12}$$

5. Katman: Üçüncü katmandaki her bir kural çıkışı, beşinci katmanda ağırlıklı toplama işlemine tabi tutulur. Böylece sonraki adımda kullanılacak dinamik geri besleme sinyali elde edilir (4.13). Her kural için bir geri besleme değişkeni bulunmaktadır. Burada; N_r kural sayısı olmak üzere (N_r^2) adet (w_{ik}) parametre bulunmaktadır.

$$h_i(t+1) = \left(\sum_{k=1}^{Nr} \mu_k(x(t), h_i(t)) \right).w_{ik} \tag{4.13}$$

6. Katman: Son katmanda ise ağırlıklı ortalama yöntemi ile durulaştırma yapılır ve ağ çıkışı elde edilir.

$$y(t+1) = \frac{\sum_{i}^{N_r} R_i.f_i}{\sum_{i}^{N_r} R_i} \tag{4.14}$$

Ağda eğitim sırasında ayarlanması gereken toplam $N_r(N_r+3n+2)$ adet parametre bulunmaktadır. Bu ağ yapısıyla ilgili daha ayrıntılı bilgi almak için [5]'den yararlanılabilinir.

4.4 RFNN2 Yapısındaki Ağ

Bu ağ yapısında geçmiş zamana ait bilgiler geri besleme döngüleriyle ağ içerisinde saklanmaktadır. Ağ yapısı Şekil 4.5'te verilmiştir [19]. Tez içersinde bu ağdan RFNN olarak bahsedilecektir. Bu ağın farklılığı; giriş değişkenlerinin, geçmişte bulanık kümelere üyelik derecelerinin geri beslenerek, girişlere eklenmesidir. Yani kural çıkışları değil, kural aktiflik derecelerini belirleyen, giriş üyelik dereceleri geri besleme yapılarak ağ içinde tutulmaktadır. Ağın her katmanındaki düğümlerin çıkışları ve eğitim sırasında ayarlanması gereken parametreler sırasıyla açıklanmıştır.

1. Katman: Ağın birinci katmanında harici giriş değişkenleri bulunmaktadır. Bu katmanda herhangi bir parametre bulunmamaktadır.

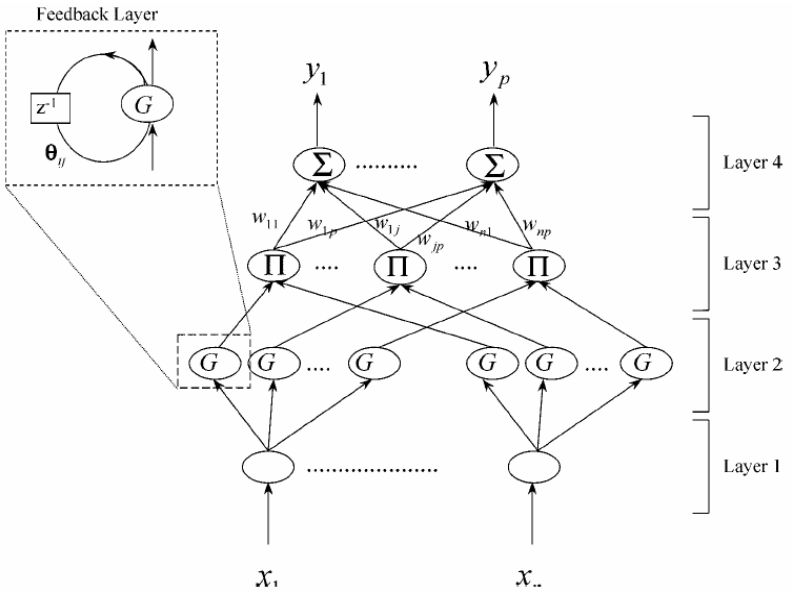
2. Katman: Bu katmanda üyelik fonksiyonlarını temsil eden düğümler bulunmaktadır. Bu düğümlerde diğer ağ yapılarında olduğu gibi Gauss biçimli üyelik fonksiyonu kullanılabilir. Bu düğümlerin çıkışı, geri besleme yapılarak, ağırlıklı olarak giriş değişkenlerine eklenmekte ve düğüm girişi elde edilmektedir.

$$u_{ij}^{2}(k) = x_i(k) + \theta_{ij}.O_{ij}^{2}(k-1) \tag{4.15}$$

u^2_{ij}, O^2_{ij}: 2 katmandaki i'ninci girişin, j'ninci bulanık kümesinin girişini ve çıkışını, θ_{ij} ise bağlantı ağırlığı parametrelerini belirtmektedir. Bu katmandaki her bir düğüm için 3 (2 ÜF parametresi, 1 bağlantı ağırlığı) parametre bulunmaktadır.

3. Katman: Üçüncü katmanda T-norm işleviyle kural çıkışları elde edilmektedir.

$$O_i^3 = \prod_i u_i^3 \text{ , (T-norm: cebirsel çarpım)} \tag{4.16}$$



Şekil 4.5: RFNN yapısındaki ağ [19]

4. Katman: Son katmanda kural çıkışlarının ağırlıklı toplamıyla çıkış değişkenleri hesaplanmaktadır. Bu katmanda her çıkış için kural sayısı kadar bağlantı ağırlığı parametresi bulunmaktadır.

$$y_j = O_i^4 = \sum_i^m u_{ij}^4 . w_i^4 \text{ , } (w^4{}_i\text{: bağlantı ağırlığı}) \qquad (4.17)$$

Bu ağ yapısıyla ilgili daha ayrıntılı bilgi almak için [19]'den yararlanılabilinir.

BÖLÜM 5: BULANIK – NÖRAL KONTROLÖR EĞİTİMİ

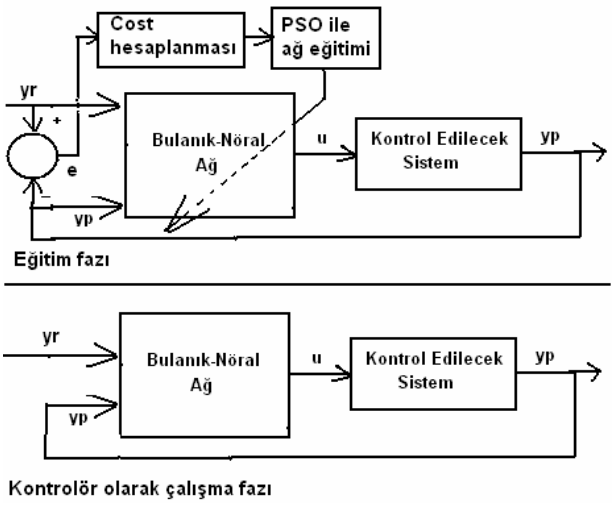
5.1 Giriş

Bu bölümde bulanık-nöral ağların kontrolör olarak eğitilmesi üzerinde durulmuştur. PSO ve HPSOGA algoritmalarıyla eğitilen, önceki bölümde TRFN olarak anlatılan yapıdaki bulanık-nöral kontrolörlerin, dinamik bir sistem üzerindeki başarımlarını görmek için yapılan benzetimler sonuçları ve algoritmaların karşılaştırılması sunulmuştur. Sistem modellemeleri, ağ eğitimleri ve benzetimler Matlab programıyla gerçekleştirilmiştir.

5.2 Bulanık – Nöral Ağın Kontrolör Olarak Eğitilmesi

Bulanık-nöral ağın kontrolör olarak eğitilmesi ve bu kontrolör ile bir sistemin kontrol edilmesine ilişkin blok şema Şekil 5.1'de verilmiştir. Bulanık-nöral ağdan beklenen, sistemi istenen ön ayar değerinde tutmak için gerekli olan kontrol sinyalini (u) üretmesidir. Bulanık-nöral ağın girişleri; y_r (set değer/ön ayar değeri) ve yp (sistemin şu anki durumu), ağ çıkışı ise sisteme uygulanacak kontrol sinyali (u) olarak tanımlanmıştır. Ağ girişleri ikişer adet Gauss biçimli üyelik fonksiyonu ile bulanıklaştırılmıştır.

Eğitim fazında; sırasıyla tüm parçacıkların temsil ettikleri ağ parametrelerine göre sistem belirli bir süre çalıştırılır ve parçacıklara ilişkin ölçüt fonksiyonları hesaplanır. Algoritmanın işletilmesiyle parçacıklar güncellenir ve bir sonraki iterasyona geçilir. Eğitim fazı belirli bir iterasyon sayısına ulaşılınca veya istenilen ölçüt fonksiyonu elde edildiğinde (sonlandırma kriteri) sonlandırılır. Eğitim fazının son iterasyonundaki en iyi ölçüt fonksiyonuna sahip parçacık (gbest), kontrolör parametrelerini temsil etmektedir. Kontrolör olarak çalışma fazında bu parametrelere göre bulanık-nöral ağ işletilir.



Şekil 5.1: Bulanık-Nöral ağ ile sistem kontrolü

Eğitimlerde PSO ve HPSOGA algoritmalarına ait aşağıda verilen parametreler kullanılmıştır. Eğitimlerde parçacıklar için herhangi bir sınırlama getirilmemiştir. Yani parçacıklar arama uzayında serbest bırakılmıştır. Çözülecek problem için bazı koşullamalar mevcutsa parçacıklar bu koşullamalara göre sınırlandırılabilir. Veya sezgisel olarak arama uzayının boyutları ve parçacıkların elde edebilecekleri en büyük ve en küçük değerler biliniyorsa, parçacıklar bu değerlerde sınırlandırılabilir.

- Parçacık sayısı=60
- Eylemsizlik ağırlığı(w)= [1,2 0,8] aralığında iterasyona bağlı azalan değer.
- Öğrenme oranları $c_1=c_2=2$
- Mutasyon olasılığı=0.1
- Çaprazlama olasılığı=0.5

Algoritmaların yapısındaki rasgelelikten dolayı bazen çok iyi, bazen de çok kötü sonuçlar alınabilmektedir. Bu yüzden; algoritmaların bulanık-nöral kontrolör eğitimindeki başarımını görebilmek için ağlar rasgele başlangıç koşulları için 100'er kez eğitilmişler ve elde edilen en iyi ve ortalama sonuçlar sunulmuştur.

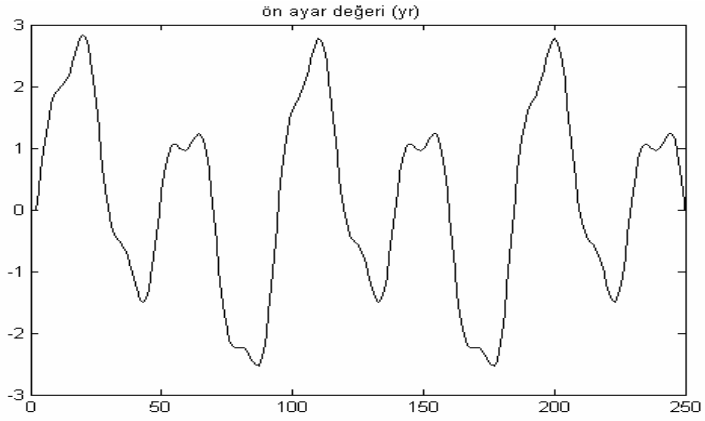
5.3 Tek Giriş, Tek Çıkışlı Ayrık Dinamik Sistem Kontrolü

Ayrık zaman ifadesi eşitlik 5.1'de verilen dinamik sistemi kontrol edebilmek için TRFN ağ yapısında PSO ve HPSOGA algoritmalarıyla bulanık-nöral kontrolör eğitimi yapılmış, iki algoritmanın başarımları karşılaştırılmıştır. Eğitimler sistemi, eşitlik 5.2'de ifade edilen oldukça değişken bir ön ayar değeri yörüngesinde tutmak için gerçekleştirilmiştir [5]. Başlangıçta sistemin $y_p[0]=0$ durumunda olduğu kabul edilmiştir.

$$y_p(k+1)=\frac{y_p(k).y_p(k-1).(y_p(k-2)+2,5)}{1+y_p^2(k)+y_p^2(k-1)}+u(k) \tag{5.1}$$

Ön ayar değeri yörüngesi Şekil 5.2'de görülmektedir. Eğitimler; yukarıda verilen parametrelerle, rasgele başlangıç değerleri için 100 kez tekrarlanmış elde edilen ortalama ve en iyi sonuçlar verilmiştir. Bu benzetimi gerçekleştiren Matlab programı ve ilgili fonksiyonları Ek-3'de verilmiştir.

$$\begin{aligned} y_r(k+1) &= 0,6.y_r(k)+0.2y_r(k-1)+0,5.\sin(2\pi\pi/45) \\ &+0,2.\sin(2\pi\pi/15)+0,2.\sin(2\pi\pi/90) \end{aligned} \tag{5.2}$$

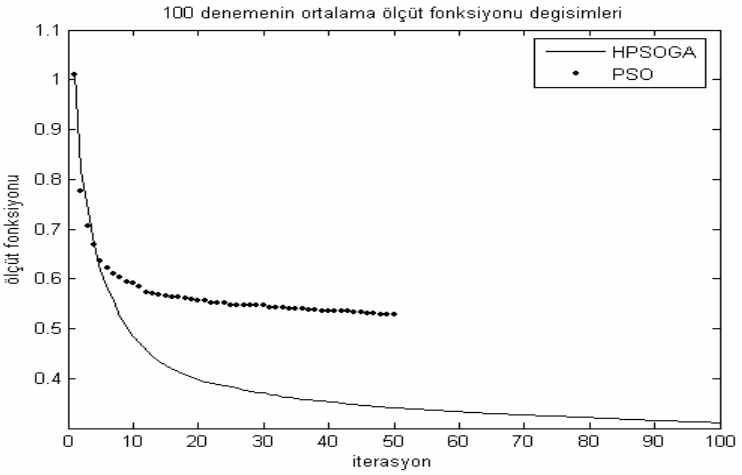


Şekil 5.2: Değişken ön ayar değeri (y_r)

Ölçüt fonksiyonu olarak eşitlik 5.3'de verilen fonksiyon kullanılmıştır. Benzetimlerde N=250 ayrık örnek değeri kullanılmıştır.

$$J=\left(\frac{1}{N}\sum_{k=1}^{N}\left(y_r(k)-y_p(k)\right)^2\right)^{1/2} \tag{5.3}$$

Rasgele başlangıç pozisyonlarında 100 kez tekrarlanan eğitimler sonucunda elde edilen ortalama ve en iyi ölçüt değerleri Tablo 5.1'de verilmiştir. Eğitimler süresince ölçüt fonksiyonlarının ortalama değişimi Şekil 5.3'de görülmektedir.



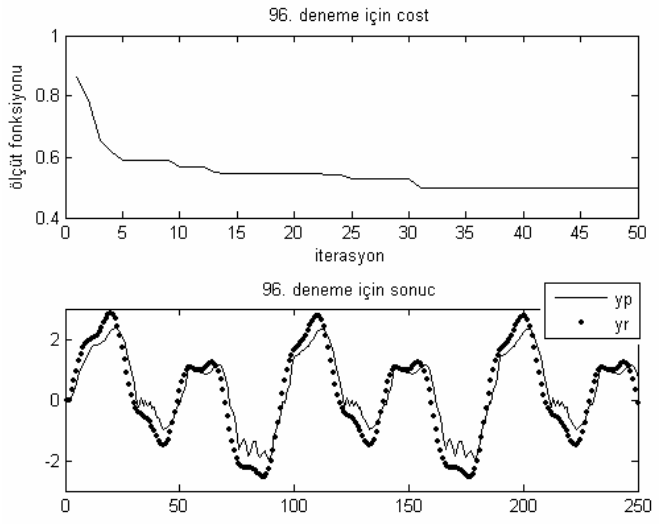
Şekil 5.3: Ortalama ölçüt fonksiyonu değişimleri

Tablo 5.1: Elde edilen en düşük ölçüt değerleri

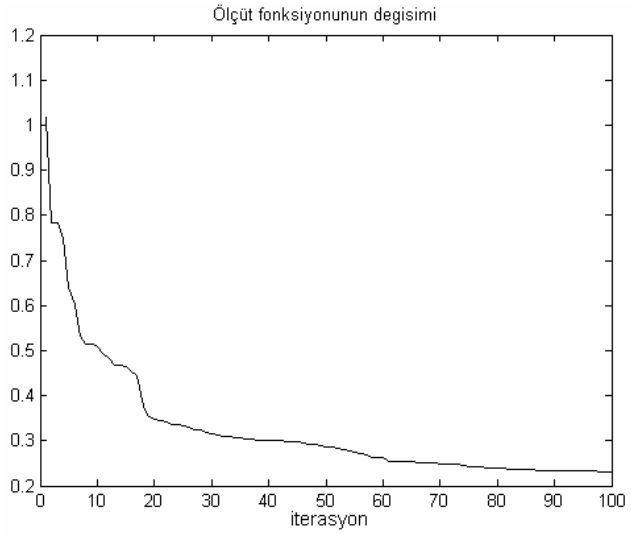
	HPSOGA	**PSO**
Ortalama Ölçüt Değeri	0,3113	0,5269
En iyi ölçüt değeri	0,2130	0,3088

HPSOGA algoritmasıyla yapılan eğitimler 100 iterasyon sürdürülmüştür. PSO algoritmasıyla yapılan eğitimler ise 50 iterasyonda sonlandırılmıştır. PSO algoritması yerel çözümlere takılabildiği için uzun iterasyon sayılarında fazla bir iyileşme sağlanamamaktadır aksine bu durum fazla işlem yüküne sebep olmaktadır. Bu duruma örnek bir iterasyonun seyri Şekil 5.4a'da görülmektedir. HPSOGA algoritmasının, yerel çözümlere takılma olasılığı, PSO algoritmasından daha az olduğu için gerçekleştirilen 100 denemenin büyük bir çoğunluğunda iyi sonuçlar elde edilmiştir. PSO algoritmasıyla gerçekleştirilen eğitimlerde ise bu oran daha düşüktür. HPSOGA algoritmasıyla gerçekleştirilen eğitimler sonucunda elde edilen kontrol sonuçlarından biri Şekil 5.5'de, bu eğitimin seyri ise Şekil 5.6'da görülmektedir. Sistem çıkışının istenilen ön ayar değeri yörüngesini oldukça iyi bir şekilde takip ettiği görülmektedir.

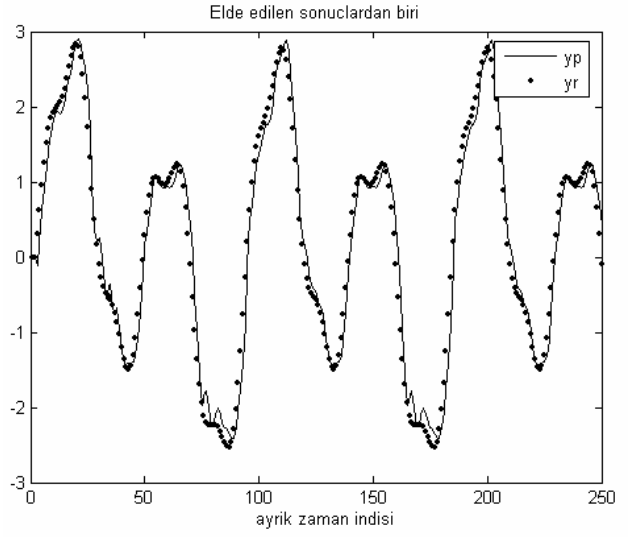
PSO algoritmasıyla gerçekleştirilen eğitimler sonucunda elde edilen kontrol sonuçlarında ise ölçüt fonksiyonunun yeterince azaltılamadığı, sistem çıkışının ön ayar değerini salınımlarla takip ettiği görülmüştür. Bu durum Şekil 5.4b'de görülmektedir.



Şekil 5.4: PSO algoritmasıyla gerçekleştirilen 96. deneme için elde edilen sonuçlar: a) eğitim boyunca ölçüt fonksiyonunun değişimi. b) eğitilen ağın kontrol başarımı

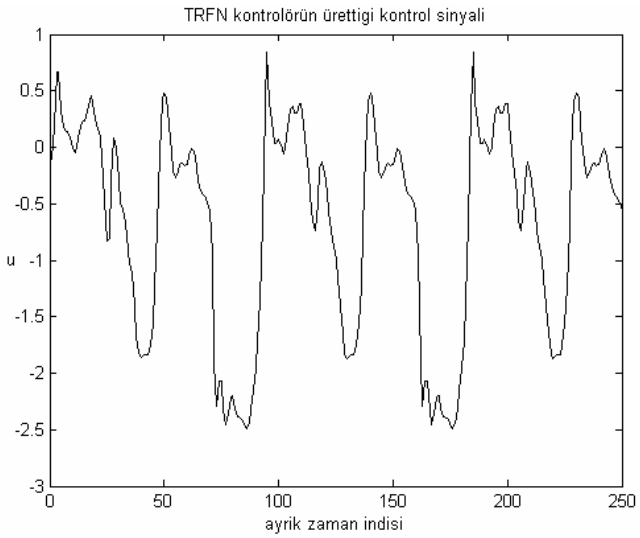


Şekil 5.5: HPSOGA algoritmasıyla gerçekleştirilen eğitim süresince ölçüt fonksiyonunun değişimi



Şekil 5.6: HPSOGA algoritmasıyla eğitilen ağla kontrol edilen sistem çıkışı

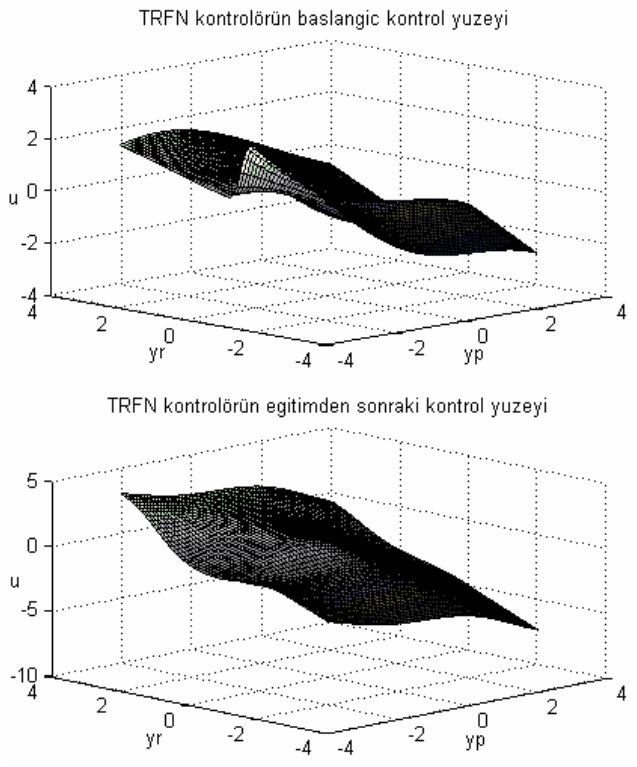
Sistemin yukarıdaki gibi bir çıkış üreterek istenilen ön ayar değerini takip edebilmesi için bulanık-nöral kontrolör tarafından üretilip, sisteme giriş olarak uygulanan kontrol sinyalinin (u) değişimi Şekil 5.7'de görülmektedir.



Şekil 5.7: TRFN yapısındaki kontrolörün ürettiği kontrol sinyali

Şekil 5.8'de ise bu benzetimde kullanılan TRFN yapısındaki bulanık-nöral ağın gerçeklediği giriş-çıkış yüzeyi verilmiştir. Eğitim fazının ilk iterasyonundaki en iyi parçacığın temsil ettiği parametreleri içeren bulanık-nöral ağın gerçeklediği kontrol yüzeyi ile, HPSOGA algoritmasıyla ile 100 iterasyon eğitildikten sonra gerçeklediği

kontrol yüzeyine bakıldığında, algoritmanın kontrolörü (bulanık-nöral ağı) ilgili uzayda bambaşka bir konuma getirdiği görülmektedir. Ön ayar değeri pozitif büyük bir değerde iken, sistem çıkışı negatif bir değerde ise, kontrolör sitem çıkışını istenilen pozitif değere çekebilmek için pozitif büyük bir kontrol sinyalini sisteme uygulayacaktır. Benzer şekilde ön ayar değeri negatif, sistem çıkışı pozitif ise, bu kez kontrolör sistem çıkışını istenen değere getirebilmek için negatif bir kontrol sinyali üretecektir. Sistem çıkışı ve ön ayar değerinin belirli bir aralıktaki değerleri için kontrolörün vereceği yanıt, aşağıdaki şekilde görüldüğü gibi yumuşak geçişli bir yüzey şeklinde olmaktadır.



Şekil 5.8: TRFN yapısındaki kontrolörün sağladığı kontrol yüzeyi (a) Başlangıçtaki yüzey (b) Eğitim sonundaki yüzey

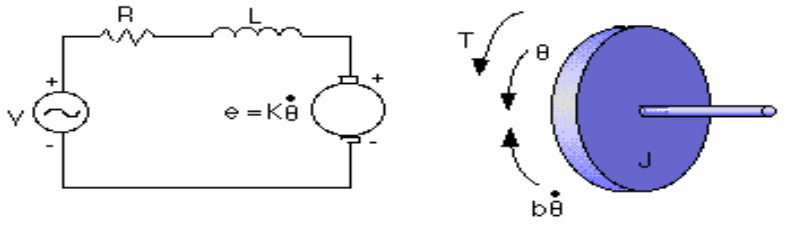
5.4 Sürekli Zamanda Benzetim Örneği için Kontrolör Eğitimi (DC Motor Hız Kontrolü)

Bu bölümde DC Motor hızını kontrol edebilmek için bulanık-nöral ağların kontrolör olarak eğitilmesi anlatılacaktır. Dördüncü bölümde anlatılan üç farklı bulanık-nöral ağ yapısındaki kontrolör HPSOGA algoritmasıyla eğitilmiş ve başarımları yapılan benzetimlerle karşılaştırılmıştır. Sistem modellemeleri, ağ eğitimleri ve benzetimler Matlab programıyla gerçekleştirilmiştir. İlgili programları Ek-4'te verilmiştir.

DC motorlar, kontrol sistemlerinde sıklıkla denetim elemanı olarak kullanılmaktadırlar. Sağladıkları dönme hareketi sırasında açısal hızlarının ve açısal konumlarının kontrol edilmesi gerekir. Bu çalışmada HPSOGA algoritmasıyla eğitilen 3 farklı bulanık-nöral ağ ile DC motor hızı kontrol edilmeye çalışılmıştır. DC motorun aşağıda verilen fiziksel parametrelere sahip olduğu varsayılarak modellenmesi gerçekleştirilmiştir [20, 21].

- Rotorun eylemsizlik momenti (J) = 0.01 $kg.m^2/s^2$
- Mekanik sistemin sönümleme sabiti (b) = 0.1 Nms
- Elektromotor kuvveti sabiti (K=Ke=Kt) = 0.01 Nm/Amp
- Stator direnci (R) = 1 ohm
- İndüktans (L) = 0.5 H

DC motorun armatürüne ait elektrik diyagramı ve rotora ait serbest cisim diyagramı Şekil 5.9'da verilmiştir.



Şekil 5.9: DC motor modeli [20, 21]

Motorun torku T, Kt sabitiyle armatür akımına(i) bağlıdır (eşitlik 5.4). Ters elektromotor kuvveti(e) ise Ke sabitiyle açısal hıza bağlıdır (eşitlik 5.5).

$$T = K_t.i \tag{5.4}$$

$$e = K_e.\dot{\theta} \tag{5.5}$$

Yukarıdaki şekilden; Newton ve Kirchoff yasalarına göre aşağıdaki eşitlikler elde edilir.

$$J.\ddot{\theta} + b.\dot{\theta} = K.i \tag{5.6}$$

$$L.\frac{di}{dt} + R.i = V - K.\dot{\theta} \tag{5.7}$$

Açısal hız ve armatür akımı durum değişkeni olarak, giriş değişkeni olarak gerilim, çıkış değişkeni olarak açısal hız alındığında DC motor aşağıdaki durum-uzay eşitlikleriyle modellenebilir. Durum-uzay denklemlerinden sistemin doğrusal bir sistem olduğu açıkça görülmektedir.

$$\frac{d}{dt}\begin{bmatrix}\dot{\theta}\\ i\end{bmatrix} = \begin{bmatrix}-\frac{b}{J} & \frac{K}{J}\\ -\frac{K}{L} & -\frac{R}{L}\end{bmatrix}.\begin{bmatrix}\dot{\theta}\\ i\end{bmatrix} + \begin{bmatrix}0\\ \frac{1}{L}\end{bmatrix}.V \tag{5.8}$$

$$\dot{\theta} = \begin{bmatrix}1 & 0\end{bmatrix}.\begin{bmatrix}\dot{\theta}\\ i\end{bmatrix} \tag{5.9}$$

Benzetimlerde, DC motora ait durum denklemleri Matlab programı yardımıyla dördüncü dereceden Runge-Kutta metodu kullanılarak çözülmüştür. Sayısal analizde Runge-Kutta yöntemleri, adi diferansiyel denklemlerin çözüm yaklaşımları için kapalı ve açık yinelemeli yöntemler ailesinin önemli bir tipidir. Bu yöntem 1900'lü yıllarda C. Runge ve M.W. Kutta adlı matematikçiler tarafından geliştirilmiştir.Bu yöntem hakkında ayrıntılı bilgi için [22, 23]' den yararlanılabilinir.

DC motor hızını kontrol edebilmek için eğitilecek bulanık-nöral ağlar, karşılaştırma yapabilmek için benzer giriş ve ÜF yapısında tasarlanmış ve HPSOGA algoritmasının aşağıda verilen parametreleriyle eğitilmiştir. Sistemin benzetimi [0 0] başlangıç koşulu altında, 5 sn boyunca işletilmiştir. Eğitimler sinüs biçimli sürekli

değişen bir ön ayar değeri için rasgele parametrelerle başlanıp, 50 kez tekrarlanmış ve elde edilen ortalama ve en iyi sonuçlar verilmiştir.

- Parçacık sayısı=60
- Eylemsizlik ağırlığı(w)= [1,2 0,8] aralığında iterasyona bağlı azalan değer.
- Öğrenme oranları $c_1=c_2=2$
- Mutasyon olasılığı=0.1
- Çaprazlama olasılığı=0.5

ANFIS yapısındaki ağda; giriş kısmında Gauss biçimli 4 adet üyelik fonksiyonu için 8 parametre, çıkış kısmında ise 4 kural için 12 adet parametre olmak üzere ağda toplam 20 parametre bulunmaktadır. Sürüdeki her bir parçacık 20 adet parametre içerecek şekilde aşağıdaki gibi düzenlenmiştir.

$$P_i=[c_1 \;\; \sigma_1 \;\; .....c_4 \;\; \sigma_4 \;\; p_1 \;\; q_1 \;\; r_1 \;\; ..... p_4 \;\; q_4 \;\; r_4] \tag{5.10}$$

TRFN yapısındaki ağda; 2 giriş için Gauss biçimli 4 üyelik fonksiyonu, 4 geri besleme girişi için de sigmoidal biçimli 4 üyelik fonksiyonu kullanılmıştır. Bu fonksiyonlar toplam 16 adet parametre içermektedir. Ağın üçüncü katmanında 4 adet kural düğümünden, 4 adet geri besleme birimine 16 adet bağlantı ağırlığı parametresi bulunmaktadır. Çıkış katmanında her bir kural için, 4 parametre (1 sabit, 2 giriş ve 1 kural çıkışı bağlantı ağırlıkları), 4 kural için toplam 16 parametre bulunmaktadır. Yani ağda ayarlanması gereken 48 adet parametre bulunmaktadır. Parçacıklar aşağıdaki gibi düzenlenmiştir.

$$P_i=[c_1 \;\; \sigma_1 \;\; .....c_4 \;\; \sigma_4 \;\; .....w_{11}w_{12}.....w_{44} \;\; a_1 \;\; a_2..... \;\; .. \;\; a_{16}] \tag{5.11}$$

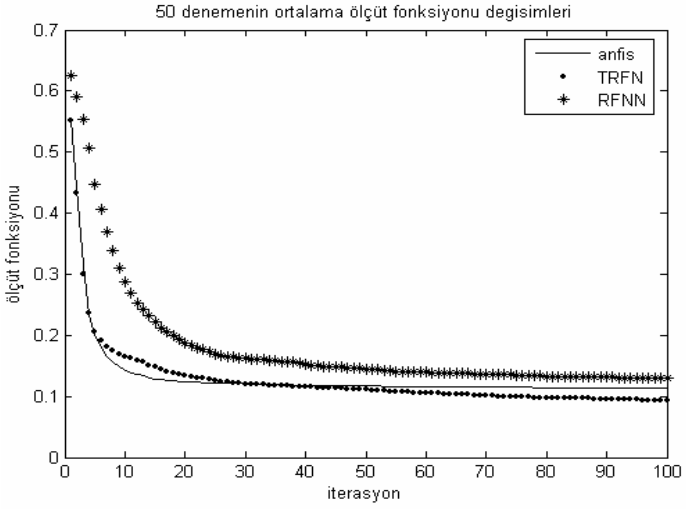
RFNN yapısındaki ağda; giriş kısmında Gauss biçimli 4 adet üyelik fonksiyonu için 8 parametre, üyelik fonksiyonlarının geri besleme bağlantı ağırlıkları için 4 parametre, çıkış katmanında 4 kural için 4 adet bağlantı ağırlığı parametresi olmak üzere toplam 16 adet parametre bulunmaktadır. Parçacıklar aşağıdaki gibi düzenlenmiştir.

$$P_i=[c_1 \;\; \sigma_1 \;\; ..... c_4 \;\; \sigma_4 \;\; \theta_1..... \;\; \theta_4 \;\; w_1 \;\; ..... \;\; w_4] \tag{5.12}$$

100 iterasyonluk ağ eğitimleri; farklı başlangıç koşulları için 50 kez tekrarlanmış, elde edilen en iyi ve ortalama ölçüt değerleri Tablo 5.2'de verilmiştir. 50 kez tekrarlanan eğitimler sonunda elde edilen sonuçlara göre TRFN yapısındaki hem ortalama hem de anlık olarak, diğer ağ yapılarından daha iyi sonuç verdiği Tablo 5.2'de görülmektedir. Farklı ağ yapıları için 50 kez tekrarlanan eğitimler sırasında ölçüt fonksiyonunun değişimlerinin ortalaması Şekil 5.10'da görülmektedir.

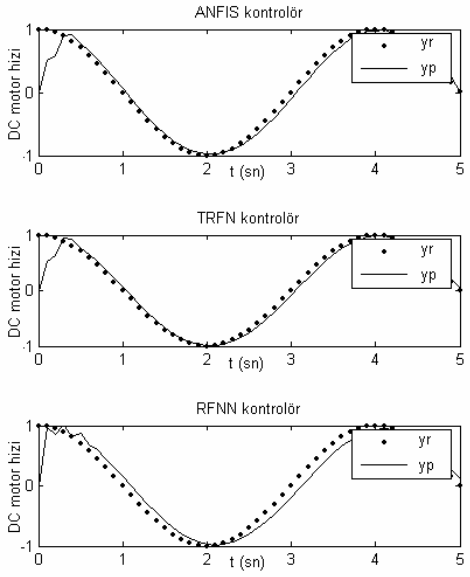
Tablo 5.2: Elde edilen en düşük ölçüt değerleri

	ANFIS	**TRFN**	**RFNN**
Ortalama Ölçüt Değeri	0,1133	0,0934	0,1295
En Düşük Ölçüt Değeri	0.1112	0,0282	0,0483

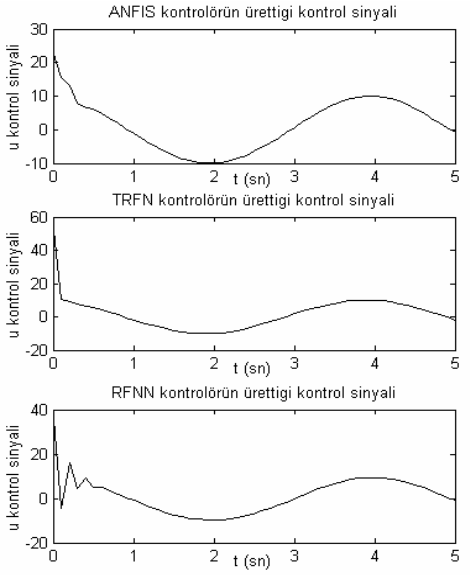


Şekil 5.10: Ağların eğitimi sırasında ortalama ölçüt fonksiyonunun değişimi

Eğitimler sonucunda sistem çıkışının yani DC motor hızının istenilen ön ayar değeri yörüngesini takip ettiği Şekil 5.11'da görülmektedir. Şekillerden de görüldüğü gibi sistem; 0.5 sn gibi kısa bir sürede, kalıcı durum hatası ve aşma gerçekleşmeden istenilen ön ayar değerine ulaşmıştır. Bu sonuçlara göre algoritmanın farklı yapılardaki kontrolör ağ parametrelerini başarıyla bulduğu görülmektedir. Şekil 5.12'de bulanık-nöral kontrolörler tarafından üretilip, sisteme giriş olarak uygulanan kontrol sinyalinin (u) değişimi görülmektedir.

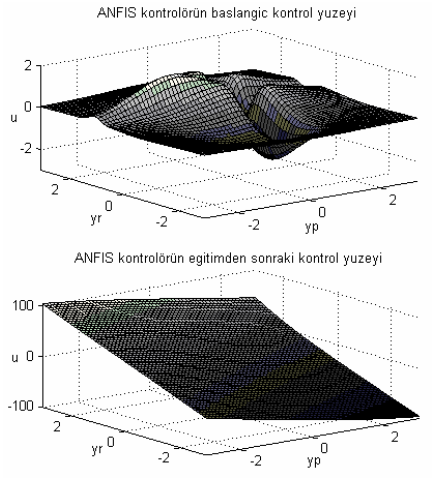


Şekil 5.11: Eğitimler sonucunda elde edilen ağ yapılarıyla kontrol edilen DC motor hızının değişimi

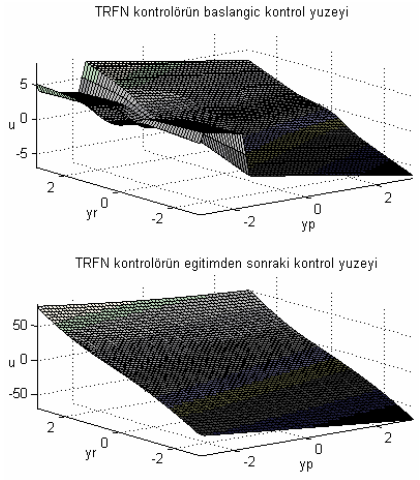


Şekil 5.12: Eğitilen bulanık-nöral kontrolörlerin ürettiği kontrol sinyali

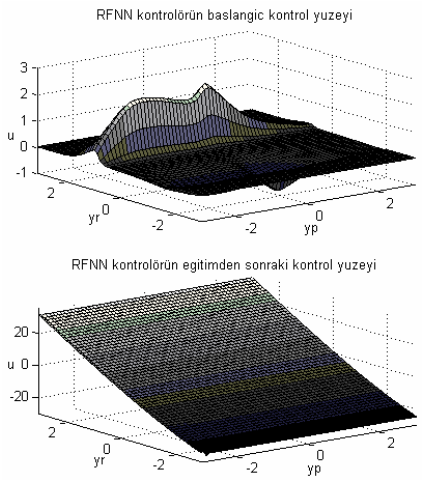
Şekil 5.13, 5.14 ve 5.15'te ise kontrolör olarak eğitilen 3 farklı yapıdaki bulanık-nöral ağların gerçeklediği giriş-çıkış (kontrol) yüzeyleri verilmiştir. Eğitim fazının ilk iterasyonu ve eğitim sonu için bulanık-nöral ağların gerçeklediği kontrol yüzeylerine bakıldığında, algoritmanın kontrolörü (bulanık-nöral ağı) ilgili uzayda bambaşka bir konuma getirdiği görülmektedir. Bulanık kontrolörler giriş değişkenlerine göre doğrusal yanıt vermektedirler.



Şekil 5.13: ANFIS yapısındaki ağın kontrol yüzeyi

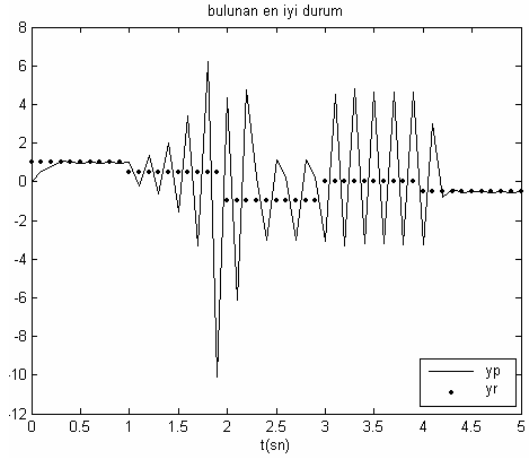


Şekil 5.14: TRFN yapısındaki ağın kontrol yüzeyi



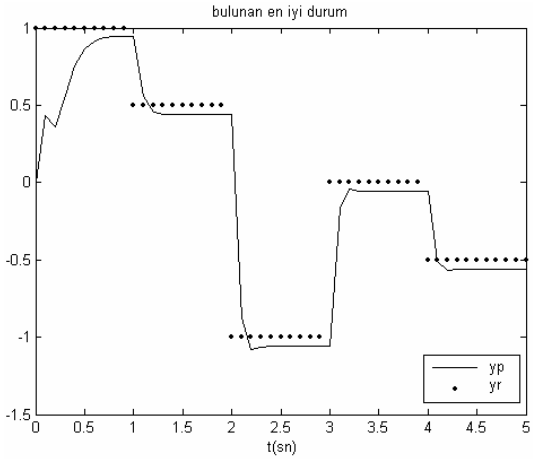
Şekil 5.15: RFNN yapısındaki ağın kontrol yüzeyi

Belirli bir aralıkta sinüzoidal biçimde değişen ön ayar değeri için eğitim yapılmasındaki amaç bu aralıkta istenilen tüm ön ayar değerleri için iyi yanıt verecek global bir bulanık-nöral kontrolör tasarlamaktı. Yukarıda eğitim sonuçları verilen bulanık-nöral ağlar ile sinüzoidal biçimde değişen ön ayar değeri için iyi kontrol sonuçları elde edilmesine rağmen, ön ayar değerinin [-1, 1] değişim aralığındaki rasgele değişen çeşitli değerler için iyi sonuçlar elde edilememiştir. TRFN yapısındaki ağ ile elde edilen sonuçlar Şekil 5.16' da görülmektedir.



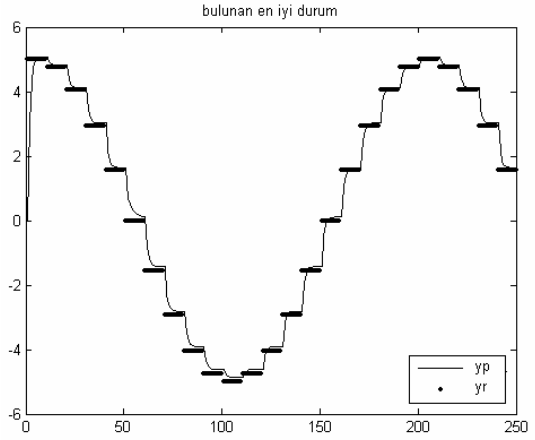
Şekil 5.16: Çeşitli ön ayar değerleri için elde edilen sonuç

Yapılan 100 iterasyonluk eğitimler ile bulanık-nöral ağların sinüs biçimli ön ayar değeri için ezberleme yapabileceği düşünülmüş ve daha sonra yapılan benzetimlerde eğitimler 10 iterasyonda sonlandırılmıştır. Bu benzetime ait sonuçlar Şekil 5.17'de görülmektedir.

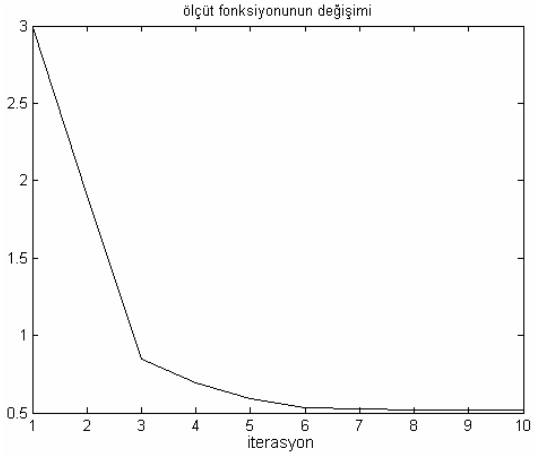


Şekil 5.17: Çeşitli ön ayar değerleri için elde edilen sonuç

Kısa süreli eğitim ile, ağın ezberleme yapmadan, çeşitli ön ayar değerleri için nispeten daha iyi sonuçlar verdiği görülmüştür. Daha sonra yapılan benzetimlerde sinüs biçimli ön ayar değerinin değişim aralığı genişletilmiştir. [-5, 5] aralığında değişen sinüs sinyalinin her ayrık değeri için daha fazla örnek alınarak, 10 iterasyonluk kısa bir eğitim yapılmıştır. Daha önce gerçekleştirilen benzetimlerde 5 sn boyunca 0.1 sn'lik aralıklarla toplam 50 adet ayrık değer için eğitim yapılıyorken, sonraki benzetimde 5 sn boyunca 0.2 sn'lik aralıklarla 25 adet ayrık değer alınmış ve her değerin 10 kez tekrarlanmasıyla 250 adet ayrık değerden oluşan basamaklı sinüzoidal bir ön ayar değeri için eğitim yapılmıştır. Bu benzetime ait elde edilen sonuçlar Şekil 5.18'de verilmiştir. Şekil 5.19'da ise eğitim sırasında ölçüt fonksiyonunun değişimi görülmektedir.

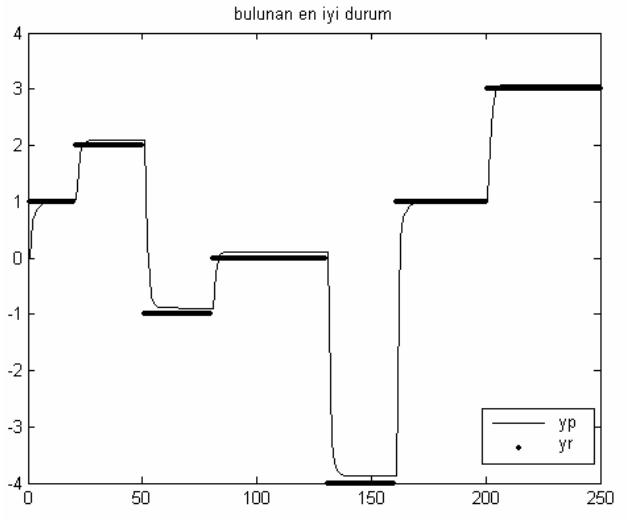


Şekil 5.18: Eğitim sonunda elde edilen durum

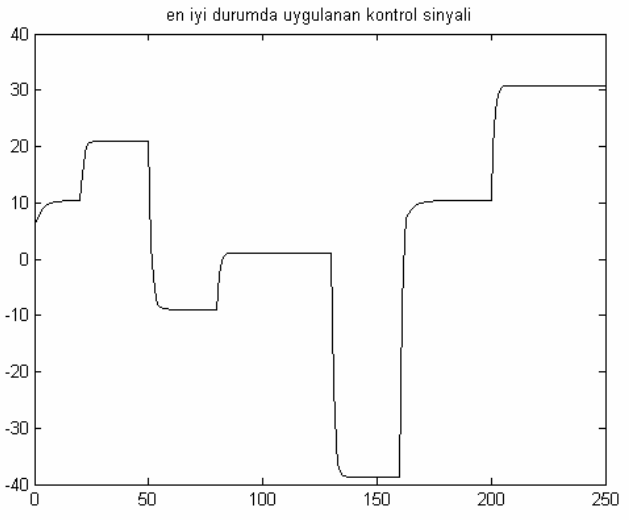


Şekil 5.19: Eğitim sırasında ölçüt fonksiyonunun değişimi

Şekil 5.18'de verilen ön ayar değeri için, Şekil 5.19'da verilen eğitim seyri sonrasında TRFN yapısındaki bulanık-nöral ağ ön ayar değerinin değişim aralığındaki rasgele çeşitli değerler için test edilmiş ve oldukça iyi kontrol sonuçları elde edilmiştir. Şekil 5.20'de gerçekleştirilen test sonucu, Şekil 5.21'de ise bulanık-nöral kontrolör tarafından üretilen kontrol sinyali görülmektedir. Elde edilen sonuçlardan başlangıçtaki amacımıza uygun kontrolörü elde ettiğimiz görülmüştür.



Şekil 5.20: Çeşitli ön ayar değerleri için elde edilen test sonucu



Şekil 5.21: Bulanık-nöral kontrolörün ürettiği kontrol sinyali

BÖLÜM 6: SONUÇ VE ÖNERİLER

Bu tez çalışmasında temel olarak; bulanık-nöral ağların, parçacık sürüsü optimizasyon algoritması ile eğitilmesi amaçlanmıştır. Bulanık-nöral ağ eğitimine geçmeden önce; parçacık sürüsü kavramının daha iyi anlaşılabilmesi ile algoritma kullanılarak içerisinde bir çok yerel minimum ve maksimum noktalar içeren "peaks" fonksiyonu üzerinde çalışılmıştır. Bölüm 2'de verilen benzetim sonuçlarına bakıldığında algoritma ile, fonksiyonun global minimum ve maksimum noktalarının bulunduğu görülmektedir. Başlangıçta fonksiyon yüzeyine rasgele serpiştirilen parçacık konumlarına ve algoritma içindeki rasgele sayılara bağlı olarak yerel çözümlerin bulunduğu da olmuştur. Ancak genel olarak algoritma ile global optimum noktalar başarıyla bulunmaktadır. Hatta başlangıçta tüm parçacıklar yerel noktalara yerleştirildiğinde bile, algoritma koşturulduktan sonra global noktalar bulunmaktadır. Peaks fonksiyonu üzerinde yapılan bu çalışma ile algoritmanın fonksiyon optimizasyonu alanında kullanılabileceği görülmüştür. Çok parametreli optimizasyon problemlerinden önce, 2 parametreden oluşan bu optimizasyon problemi ile, parçacık sürüsü, global çözüm, yerel çözüm gibi kavramlar daha iyi anlaşılmıştır.

Daha sonra 2 katmanlı bir yapay sinir ağı parçacık sürüsü algoritması ile eğitilerek Exor problemi çözülmüştür. Bu problemde ayarlanması gereken 9 adet parametre vardır ve çok parametreli problemlerin optimizasyonu için güzel bir örnektir. Bölüm 2'de verilen benzetim sonuçlarına bakıldığında algoritmanın, ağ eğitimlerinde kullanılan klasik yöntemlere göre daha hızlı sonuçlar verdiği görülmektedir. PSO algoritmasının, klasik yöntemlerden en önemli farklılığı türev bilgisi içermemesidir. Ayrıca klasik yöntemlerde, başlangıç noktasına bağlı olarak çözüme çok geç ulaşılabilir veya hiç ulaşılmayabilir. Ancak parçacık sürüsü algoritması, rasgele seçilen bir çok aday çözümle başladığı için böyle bir durumla karşılaşılma olasılığı daha azdır.

Yapılan bu çalışmalardan sonra, bu tez çalışmasının asıl konusunu oluşturan bulanık-nöral ağların PSO algoritmasıyla eğitilmesine ilişkin çalışmalar yapılmıştır. Eğitilen bulanık-nöral ağların kontrolör olarak kullanılması düşünülmüş ve iki sistem üzerinde benzetimler yapılmıştır. Kontrol edilecek sistemin modelinin bilinmediği veya eğiticili öğrenme ile kontrolör tasarımı için eğitim verisinin bulunmadığı durumlarda, rasgele arama algoritmaları, özellikle türev bilgisine ihtiyaç duymadıkları için klasik ağ eğitim yöntemlerine göre üstünlük sağlamaktadır.

Tek giriş-tek çıkışlı dinamik bir sistemi istenilen ön ayar değerinde tutmak için yinelemeli bir bulanık-nöral ağ eğitimi PSO algoritması ve karma arama algoritması ile gerçekleştirilmiştir. Yapılan bu çalışmanın amacı iki algoritmanın başarımlarını karşılaştırmaktır. Tablo 5.1'de verilen benzetim sonuçlarına göre, 100 defa tekrarlanan eğitimler sonucunda karma algoritmanın, klasik PSO algoritmasından daha iyi sonuçlar verdiği görülmüştür. Karma algoritmanın yerel çözümlere takılma olasılığı PSO algoritmasından daha azdır. Benzetim sonuçlarına bakıldığında dinamik sistem için oldukça iyi kontrol sonuçları elde edildiği görülmektedir.

Daha sonra yapılan çalışma da ise, tez içersinde anlatılan 3 farklı bulanık-nöral ağ karma algoritma ile kontrolör olarak eğitilmiş ve DC motor hızını kontrol etmek için benzetimler yapılmıştır. 50 kez tekrarlanan eğitimler sonucunda, Bölüm 5.4'te verilen başarılı kontrol sonuçları elde edilmiştir. DC motor hızının değişken ön ayar değeri için verdiği yanıtlara bakıldığında, sistemin aşma ve kalıcı durum hatası meydana gelmeden 0.5 sn gibi kısa bir sürede istenen değere oturduğu görülmektedir. Başlangıçta rasgele parametreler içeren bulanık-nöral ağların gerçekledikleri kontrol yüzeyleri ile, HPSOGA algoritmasıyla eğitildikten sonra gerçekledikleri kontrol yüzeylerine bakıldığında, algoritmanın kontrolörleri ilgili uzayda bambaşka bir konuma getirdiği görülmektedir.

Tablo 5.2'de verilen sonuçlara bakıldığında TRFN yapısındaki ağ ile, hem ortalama hem de anlık (gerçekleştirilen bir deneme) olarak diğer ağlarda daha iyi sonuçlar alındığı görülmektedir. ANFIS yapısındaki ağ ile RFNN yapısındaki ağdan ortalama olarak daha iyi sonuçlar alınmasına rağmen, anlık olarak daha kötü sonuçlar alınmıştır.

Bu çalışmanın amacı farklı yapıdaki bulanık-nöral ağların başarımlarını karşılaştırmaktır. Ancak ağ yapılarının birbirlerine karşı üstünlüklerini test etmek için başka sistemler üzerinde, daha kapsamlı çalışmalar yapılması gerekmektedir.

Bu tez kapsamında yapılan çalışmalarla bulanık-nöral ağların parçacık sürüsü algoritması ile eğitilebileceği görülmüştür. PSO algoritması ile klasik yöntemlerden daha hızlı sonuç alınabilmektedir. Burada algoritmanın paralel arama kabiliyeti ön plana çıkmaktadır. Özellikle FPGA gibi paralel işlem yapma kabiliyetindeki işlemciler ile algoritma kullanılarak bulanık-nöral ağ eğitimleri gerçekleştirilebilir. Hesaplamalarda; algoritmanın klasik yöntemlere göre işlem yükü daha azdır. Ancak çözülecek problemdeki parametre sayısına da bağlı olmakla birlikte, algoritmada kullanılan parçacık sayısı, bellek miktarı gereksinimini artırabilmektedir. Optimal programlama becerisi ile bu sorunun aşılabileceği düşünülmektedir.

KAYNAKLAR

[1] Kennedy, J. ve Eberhart, R. C. "Particle swarm optimization", ***Proc. IEEE int'l conf. on neural Networks***, Vol. IV, s: 1942-1948. IEEE service center, Piscataway, NJ, 1995.

[2] Shi, Y. ve Eberhart, R. C. "A modified particle swarm optimizer", ***Proceedings of the IEEE International Conference on Evolutionary Computation*** s: 69-73. IEEE Press, Piscataway, NJ, 1998

[3] Shi, Y. ve Eberhart, R. C., "Parameter selection in particle swarm optimization", ***Evolutionary Programming VII***: Proc. EP 98 s: 591-600. Springer-Verlag, New York, 1998.

[4] Zhang Li-ping, Yu Huan-jun ve HU Shang-xu, "Optimal choice of parameters for particle swarm optimization", ***Journal of Zhejiang University Science***

[5] Chia-Feng Juang, "A Hybrid of Genetic Algorithm and Particle Swarm Optimization for Recurrent Network Design", ***IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics***, Vol. 34, No:2, April 2004

[6] S. He, J. Y. Wen, E. Prempain, Q. H. Wu, J. Fitch, S. Mann, "An Improved Particle Swarm Optimization for Optimal Power Flow", ***International Conf. on Power System Technology***, November 2004

[7] M. Aliyari Shoorehdeli, M. Teshnehlab, A. K. Sedigh, "A Novel Training Algorithm in ANFIS Structure", ***Proceedings of the 2006 American Control Conference Minneapolis***, Minnesota, USA

[8] A. A. A. Esmin, G. Lambert-Torres, "Fitting Fuzzy Membership Functions using Hybrid Particle Swarm Optimization", ***2006 IEEE International Conference on Fuzzy Systems***, July 16-21, 2006

[9] Fuqing Zhao, Zongyi Ren, Dongmei Yu, Yahong Yang, "Application of An Improved Particle Swarm Optimization Algorithm for Neural Network Training", ***0-7803-9422-4/05/2005 IEEE***

[10] C. F. Juang, Chao-Hsin Hsu, "Temperature Control by Chip-Implemented Adaptive Recurrent Fuzzy Controller Designed by Evolutionary Algorithm", ***IEEE Transactions on Circuits ans Systems-1: Regular Papers***, Vol.52, No.11, November 2005

[11] C. F. Juang, C. F. Lu, "Load-frequency control by hybrid evolutionary fuzzy PI controller", ***IEE Proc.-Gener. Transm. Distrib***, Vol. 153, No:2, March 2006

[12] http://www.swarmintelligence.org/tutorials.php **(Ziyaret tarihi: 01/06/07)**

[13] S. P. Ghoshal, "Optimizations of PID gains by particle swarm optimizations in fuzzy based automatic generation control", ***Electric Power Systems Research***, Volume 72, Issue 3, s: 203-212, 15 December 2004

[14] Jih-Gau Juang, Bo-Shian Lin, Kuo-Chih Chin, "Automatic Landing Control Using Particle Swarm Optimisation", ***Proceedings of the IEEE International Conference on Mechatronics***, July 10-12 2005, Taipei, Taiwan

[15] Hamdi A. Awad, "A Novel Particle Swarm-Based Fuzzy Control Scheme", ***IEEE International Conference on Fuzzy Systems***, Canada July 16-21, 2006

[16] Karaboğa, D., 2004, "Yapay Zeka Optimizasyon Algoritmaları", ***Atlas Yayınları***

[17] http://www.mmo.org.tr/muhendismakina/arsiv/2001/ekim/Genetik_Algoritma.htm **(Ziyaret tarihi: 01/06/07)**

[18] Jyh-Shing Roger Jang, "ANFIS : Adaptive-Network-Based Fuzzy Inference System", ***Transactions on Systems, Man, and Cybernetics***, Vol. 23, No: 3, May/June 1993

[19] Ching-Hung Lee ve Ching-Cheng Teng, "Identification and Control of Dynamic Systems Using Recurrent Fuzzy Neural Networks", ***IEEE Transactions on Fuzzy Systems***, Vol. 8, No: 4, August 2000

[20] http://www.library.cmu.edu/ctms/ **(Ziyaret tarihi: 01/06/07)**

[21] http://www.engin.umich.edu/group/ctm/examples/motor/motor.html **(Ziyaret tarihi: 01/06/07)**

[22] http://math.fullerton.edu/mathews/n2003/RungeKuttaMod.html **(Ziyaret tarihi: 01/06/07)**

[23] http://en.wikipedia.org/wiki/Runge-Kutta_methods, http://tr.wikipedia.org/wiki/Runge-Kutta_Y%C3%B6ntemleri **(Ziyaret tarihi: 01/06/07)**

EK-1A

Peaks fonksiyonunun global minimum noktası bulan Matlab programı

```
clear all; close all; clc;
%% Çözmek istediğimiz fonksiyon
x1=-4:0.1:4;
x2=-4:0.1:4;
for i=1:1:81
   for j=1:1:81
      y(i,j)=3*(1-x1(i))^2*exp(-x1(i)^2-(x2(j)+1)^2)...
         -10*((1/5)*x1(i)-x1(i)^3-x2(j)^5)*exp(-x1(i)^2-x2(j)^2)...
         -(1/3)*exp(-(x1(i)+1)^2-x2(j)^2);
   end
end
subplot(2,1,1); surf(x1,x2,y)
xlabel('x2 girişi'); ylabel('x1 girişi'); zlabel('çıkış')
title('matlab peaks Fonksiyonu'); hold on;
p=10;               %% parçacık sayısı
P=randn(p,2);     %% başlangıçta hepsi rasgele atandı
% P=[-1.5 0.5;    %% baslangıçta hepsi local minimumda
%    -1.5 -0.5;
%    -1.5 0;
%    -1.3 0.5;
%    -1.3 -0.5;
%    -1.3 0;
%    -1.7 0.5;
%    -1.7 -0.5;
%    -1.7 0;
%    -1.4 0.5];
P0=P; pbest=P;
c1=2; c2=2;       %% Öğrenme faktörleri
%% Her particle için başlangıç y hesaplanıyor
%% hesaplanan en küçük y değerleri Eskiy
for n=1:1:p
   y(n)=3*(1-pbest(n,1))^2*exp(-pbest(n,1)^2-(pbest(n,2)+1)^2)...
      -10*((1/5)*pbest(n,1)-pbest(n,1)^3-pbest(n,2)^5)...
      *exp(-pbest(n,1)^2-pbest(n,2)^2)-...
      (1/3)*exp(-(pbest(n,1)+1)^2-pbest(n,2)^2);
   Eskiy(n)=y(n);
end
subplot(2,1,2); plot3(P(:,2),P(:,1),Eskiy,'k*');hold on;
Vm=randn(p,2);    %% parametreler için değişim miktarı
gbest=randn(1,2); %% pbestlerin en iyisi
```

```
w=0.9;
for k=1:1:100    %% k iterasyon sayacı en fazla 100 iterasyon
   w=w-0.009;
   for n=1:1:p    %% p=10 tane particle var
      y(n)=3*(1-P(n,1))^2*exp(-P(n,1)^2-(P(n,2)+1)^2)...
         -10*((1/5)*P(n,1)-P(n,1)^3-P(n,2)^5)*exp(-P(n,1)^2-P(n,2)^2)...
         -(1/3)*exp(-(P(n,1)+1)^2-P(n,2)^2);
      if (y(n)<Eskiy(n))
         Eskiy(n)=y(n);
         pbest(n,:)=P(n,:);
      end
   end
   for n=1:1:p
      [a b]=min(Eskiy);
      gbest(k,:)=pbest(b,:);
      pb=sum(pbest)/p;
      Vm(n,:)=w*Vm(n,:)+c1*rand*(pb-P(n,:))+c2*rand*(gbest(k,:)-P(n,:));
      P(n,:)=P(n,:)+Vm(n,:);
   end
end
x1=gbest(k,1)
x2=gbest(k,2)
y=3*(1-x1)^2*exp(-x1^2-(x2+1)^2)-10*((1/5)*x1-x1^3-x2^5)*exp(-x1^2-x2^2)...
      -(1/3)*exp(-(x1+1)^2-x2^2)

subplot(2,1,2);
plot3(P(:,2),P(:,1),Eskiy,'ro'); axis([-4 4 -4 4 -10 10]);
xlabel('x2 girişi')
ylabel('x1 girişi')
```

EK-1B

Peaks fonksiyonunun global maksimum noktası bulan Matlab programı

```
clear all; close all; clc;
%% Çözmek istediğimiz fonksiyon
x1=-4:0.1:4;
x2=-4:0.1:4;
for i=1:1:81
   for j=1:1:81
      y(i,j)=3*(1-x1(i))^2*exp(-x1(i)^2-(x2(j)+1)^2)...
         -10*((1/5)*x1(i)-x1(i)^3-x2(j)^5)*exp(-x1(i)^2-x2(j)^2)...
         -(1/3)*exp(-(x1(i)+1)^2-x2(j)^2);
   end
end
subplot(2,1,1); surf(x1,x2,y)
xlabel('x2 girişi'); ylabel('x1 girişi'); zlabel('çıkış')
title('matlab peaks Fonksiyonu'); hold on;
p=10;               %% parçacık sayısı
P=randn(p,2);     %% başlangıçta hepsi rasgele atandı
% P=[-1.5 0.5;    %% baslangıçta hepsi local minimumda
%    -1.5 -0.5;
%    -1.5 0;
%    -1.3 0.5;
%    -1.3 -0.5;
%    -1.3 0;
%    -1.7 0.5;
%    -1.7 -0.5;
%    -1.7 0;
%    -1.4 0.5];
P0=P; pbest=P;
c1=2; c2=2;       %% Öğrenme faktörleri
%% Her particle için başlangıç y hesaplanıyor
%% hesaplanan en küçük y değerleri Eskiy
for n=1:1:p
   y(n)=3*(1-pbest(n,1))^2*exp(-pbest(n,1)^2-(pbest(n,2)+1)^2)...
      -10*((1/5)*pbest(n,1)-pbest(n,1)^3-pbest(n,2)^5)...
      *exp(-pbest(n,1)^2-pbest(n,2)^2)-...
      (1/3)*exp(-(pbest(n,1)+1)^2-pbest(n,2)^2);
   Eskiy(n)=y(n);
end
subplot(2,1,2); plot3(P(:,2),P(:,1),Eskiy,'k*');hold on;
Vm=randn(p,2);    %% parametreler için değişim miktarı
gbest=randn(1,2); %% pbestlerin en iyisi
w=0.9;
for k=1:1:100    %% k iterasyon sayacı en fazla 100 iterasyon
   w=w-0.009;
   for n=1:1:p    %% p=10 tane particle var
```

```
        y(n)=3*(1-P(n,1))^2*exp(-P(n,1)^2-(P(n,2)+1)^2)...
            -10*((1/5)*P(n,1)-P(n,1)^3-P(n,2)^5)*exp(-P(n,1)^2-P(n,2)^2)...
            -(1/3)*exp(-(P(n,1)+1)^2-P(n,2)^2);
        if (y(n)>Eskiy(n))
            Eskiy(n)=y(n);
            pbest(n,:)=P(n,:);
        end
    end
    for n=1:1:p
        [a b]=max(Eskiy);
        gbest(k,:)=pbest(b,:);
        pb=sum(pbest)/p;
        Vm(n,:)=w*Vm(n,:)+c1*rand*(pb-P(n,:))+c2*rand*(gbest(k,:)-P(n,:));
        P(n,:)=P(n,:)+Vm(n,:);
    end
end
x1=gbest(k,1)
x2=gbest(k,2)
y=3*(1-x1)^2*exp(-x1^2-(x2+1)^2)-10*((1/5)*x1-x1^3-x2^5)*exp(-x1^2-x2^2)...
        -(1/3)*exp(-(x1+1)^2-x2^2)

subplot(2,1,2);
plot3(P(:,2),P(:,1),Eskiy,'ro'); axis([-4 4 -4 4 -10 10]);
xlabel('x2 girişi')
ylabel('x1 girişi')
```

EK-2A

İki katmanlı yapay sinir ağını, Exor problemini çözmek için, PSO algoritmasıyla eğiten Matlab programı

```
clear all; close all; clc;
X=[0 0 1 1;       %% Eğitim seti
  0 1 0 1];      %% Giriş çiftleri ve
yd=[0 1 1 0];     %% istenen çıkış
p=100;             %% particle sayısı
P=randn(p,9);   %% başlangıç particles
pbest=P;         %% Her particle için en küçük E değerinin elde edildiği konum
c1=2; c2=2;       %% Öğrenme faktörleri
%%Baslangıç E değerleri hesaplanıyor
%%Her particle için hesaplanan en küçük E değeri EskiE
   for n=1:1:p    %% p=30 tane particle var
      E(n)=0;
      for i=1:1:4     %% 4 örneğin ağa göstermesi
         W1=[P(n,1) P(n,2);
            P(n,3) P(n,4)];
         W2=[P(n,7) P(n,8)];
         esik1=[P(n,5);
               P(n,6)];
         esik2=P(n,9);
         %% İleri hesaplama
         Net=W1*X(:,i)+esik1;
         v=1./(ones(size(Net))+exp(-Net));
         Net=W2*v+esik2;
         y(i,n)=1/(1+exp(-Net));
         hata(i)=yd(i)-y(i,n);
         E(n)=E(n)+(1/2)*(hata(i)^2);
      end
      EskiE(n)=E(n);
   end
Vm=randn(p,9);    %% parametreler için değişim miktarı
gbest=randn(1,9); %% pbestlerin en iyisi
alfamax=1.2;
alfamin=0.8;
itmax=100;
t=0;
for k=1:1:100    %% k iterasyon sayacı en fazla 100 iterasyon
   alfa=alfamin+((alfamax-alfamin)/itmax)*(itmax-k);
   for n=1:1:p    %% p=30 tane particle var
      E(n)=0;
      for i=1:1:4     %% 4 örneğin ağa göstermesi
         W1=[P(n,1) P(n,2);
            P(n,3) P(n,4)];
         W2=[P(n,7) P(n,8)];
```

```
            esik1=[P(n,5);
                  P(n,6)];
            esik2=P(n,9);
            %% İleri hesaplama
            Net=W1*X(:,i)+esik1;
            v=1./(ones(size(Net))+exp(-Net));
            Net=W2*v+esik2;
            y(i,n)=1/(1+exp(-Net));
            hata(i)=yd(i)-y(i,n);
            E(n)=E(n)+(1/2)*(hata(i)^2);
        end
        if (E(n)<EskiE(n))
            EskiE(n)=E(n);
            pbest(n,:)=P(n,:);
        end
    end
    for n=1:1:p
        [a b]=min(EskiE);
        A(k)=a;
        gbest=pbest(b,:);
        if (a<=0.001)
          t=1; break
        end
        Vm(n,:)=alfa*Vm(n,:)+c1*rand*(pbest(n,:)-P(n,:))+c2*rand*(gbest-P(n,:));
        P(n,:)=P(n,:)+Vm(n,:);
    end
    k
    plot(A,'r'); drawnow;
    xlabel('iterasyon')
    ylabel('cost')
    title('costun iterasyona bağlı değişimi (PSO)')
    if (t==1)
      break
    end
end
yd
y(:,b)'
a
```

EK-2B

İki katmanlı yapay sinir ağını, Exor problemini çözmek için, klasik yöntemlerle eğiten Matlab programı

```
%% EXOR çözmek için klasik yöntemlerle MLP eğitimi
clear all; close all; clc;
x=[0 0 1 1;
   0 1 0 1];
yd=[0 1 1 0];
net=newff(minmax(x),[2 1],{'logsig','logsig'},'traingd');
% net=newff(minmax(x),[2 1],{'logsig','logsig'},'traingdm');
% net=newff(minmax(x),[2 1],{'logsig','logsig'},'traingda');
% net=newff(minmax(x),[2 1],{'logsig','logsig'},'traingdx');
% net=newff(minmax(x),[2 1],{'logsig','logsig'},'trainrp');
% net=newff(minmax(x),[2 1],{'logsig','logsig'},'traincgf');
% net=newff(minmax(x),[2 1],{'logsig','logsig'},'traincgp');
% net=newff(minmax(x),[2 1],{'logsig','logsig'},'traincgb');
% net=newff(minmax(x),[2 1],{'logsig','logsig'},'trainscg');
% net=newff(minmax(x),[2 1],{'logsig','logsig'},'trainbfg');
% net=newff(minmax(x),[2 1],{'logsig','logsig'},'trainoss');
% net=newff(minmax(x),[2 1],{'logsig','logsig'},'trainlm');

W1=[-1.5804   -0.6817;  % başlangıç parametreleri
   -0.0787   -1.0246];
W2=[-1.2344    0.2888];
b1=[-0.4293;
   0.0558];
b2=-0.3679;
net.iw{1}=W1;
net.lw{2}=W2;
net.b{1}=b1;
net.b{2}=b2;

net.trainParam.epochs=3000;
net.trainParam.show = 100;
net.trainParam.lr=0.5;
net.trainParam.mc = 0.5;
net.trainParam.goal=0.001;
[net,tr]=train(net,x,yd);
y = sim(net,x)
```

EK-3

HPSOGA algoritması ile eğitilen, TRFN yapısındaki bulanık-nöral kontrolör ile ayrık dinamik sistem kontrol benzetimini gerçekleştiren program ve ilgili fonksiyonlar

Gauss üyelik fonksiyonu:

```
function output=gauss(x,c,s)
output=exp(-0.5*((x-c)/s).^2);
```

Sigmoidal üyelik fonksiyonu:

```
function output=sigma1(x,c,slope)
output=1./(1+exp(-slope*(x-c)));
```

Parçacıkları uygunluk değerlerine göre iyiden kötüye doğru sıralayan fonksiyon:

```
function [P pbest Vp EskiJ]=parcacik_sirala(P,pbest,Vp,EskiJ)
temp1=[]; temp2=[]; temp3=[]; temp4=[];
for say2=1:1:size(EskiJ,1)-1
   say3=0;
   for say1=1:1:size(EskiJ,1)-1
      if EskiJ(say1)>EskiJ(say1+1)
         temp1=EskiJ(say1);
         EskiJ(say1)=EskiJ(say1+1);
         EskiJ(say1+1)=temp1;
         temp2=pbest(say1,:);
         pbest(say1,:)=pbest(say1+1,:);
         pbest(say1+1,:)=temp2;
         temp3=P(say1,:);
         P(say1,:)=P(say1+1,:);
         P(say1+1,:)=temp3;
         temp4=Vp(say1,:);
         Vp(say1,:)=Vp(say1+1,:);
         Vp(say1+1,:)=temp4;
         say3=say3+1;
      end
   end
   if say3==0
      break
   end
end
```

Sıralanmış paracıkların kötü olanlarını sürüden atıp, yerlerine çaprazlama ve değişim operatörleri ile yeni bireyler üreten fonksiyon:

```
function [P pbest Vp EskiJ]=klasikGA3(P,pbest,Vp,EskiJ)
s1=size(P,1)/2;
s2=size(P,2);
g=zeros(s1/2,2);
d=zeros(s1/2,2);
pr=0.5; %% crossover oranı
for say=1:1:s1/2
   while(g(say,1)==g(say,2))
      temp=(rand(1,s1)<pr).*rand(1,s1);
      for say1=1:1:s1
```

```
            if temp(say1)==0
               temp(say1)=1;
            end
         end
         [tmp1 tmp2]=min(temp);
         g(say,1)=tmp2;
         temp=(rand(1,s1)<pr).*rand(1,s1);
         for say1=1:1:s1
            if temp(say1)==0
               temp(say1)=1;
            end
         end
         [tmp1 tmp2]=min(temp);
         g(say,2)=tmp2;
   end
   g(say,:)=sort(g(say,:),2);
   d(say,1)=fix(15*rand+1);
   d(say,2)=fix(31*rand+17);
end
for say=1:1:s1/2
   g1=g(say,1); g2=g(say,2);
   d1=d(say,1); d2=d(say,2);
   temp1=[P(g1,1:1:d1) P(g2,d1+1:1:d2) P(g1,d2+1:1:s2)];
   temp2=[P(g2,1:1:d1) P(g1,d1+1:1:d2) P(g2,d2+1:1:s2)];
   P(s1+2*say-1,:)=temp1;
   P(s1+2*say,:)=temp2;
end
%% crossoverla elde edilen yeni parçacıklara mutasyon uygulanacak
gbest=pbest(1,:);
mr=0.1;
for say=s1+1:1:s1*2
   temp=rand(1,s2)<mr;
   for say1=1:1:size(temp,1)
      if temp(say1)==1
         P(say,say1)=randn;
      end
   end
end
%% yeni parcaciklarin başlangic degerleri atanıyor
for say=(size(P,1)/2)+1:1:size(P,1)
   pbest(say,:)=P(say,:);
   Vp(say,:)=zeros(1,size(P,2));
   EskiJ(say,:)=1000*ones(1,1);
end
```

Sürüdeki iyi uygunluk değerine sahip parçacıkları PSO algoritması ile güncelleyen fonksiyon:

```
function [P pbest gbest Vp]=klasikPSO(w,P,pbest,Vp)
c1=2; c2=2; %% ögrenme oranları
s1=size(P,1)/2;
s2=size(P,2);
gbest=pbest(1,:);
for say=1:1:s1
   a1=rand; a2=rand;
   Vp(say,:)=w*Vp(say,:)+c1*a1.*(pbest(say,:)-P(say,:))+ c2*a2.*(gbest-P(say,:));
   P(say,:)=P(say,:)+Vp(say,:);
end
```

TRFN yapısındaki bulanık-nöral ağ işleyişini gerçekleştiren fonksiyon:

```
function [kont hy]=TRFNfuzzy(yr1,yp1,h,W,cx,sx,ax)
%% yr1: set deger
%% yp1: sistemin su andaki durumu
%% h: sistemin iç dinamikleri
%% yr1 ÜF tanımları ve UF vektörü
Nyr1=gauss(yr1,cx(1),sx(1));
Pyr1=gauss(yr1,cx(2),sx(2));
yr_UF=[Nyr1 Pyr1];
%% yp1 ÜF tanımları ve UF vektörü
Nyp1=gauss(yp1,cx(3),sx(3));
Pyp1=gauss(yp1,cx(4),sx(4));
yp_UF=[Nyp1 Pyp1];
%% h ÜF vektörü
h1=sigma1(h(1),cx(5),sx(5));
h2=sigma1(h(2),cx(6),sx(6));
h3=sigma1(h(3),cx(7),sx(7));
h4=sigma1(h(4),cx(8),sx(8));
h_UF=[h1 h2 h3 h4];
%% Kural aktiflikleri (4 kural) Tnorm=ceb.carpım
r=[];
for k=1:1:size(yr_UF,2)
    for l=1:1:size(yp_UF,2)
       ra=yr_UF(k)*yp_UF(l);
       r=[r ra];
    end
end
R=r.*h_UF;  % yeni h değerleri
hy=R*W;
f1=ax(1)+ax(2)*yr1+ax(3)*yp1+ax(4)*h(1);
f2=ax(5)+ax(6)*yr1+ax(7)*yp1+ax(8)*h(2);
f3=ax(9)+ax(10)*yr1+ax(11)*yp1+ax(12)*h(3);
f4=ax(13)+ax(14)*yr1+ax(15)*yp1+ax(16)*h(4);
f=[f1 f2 f3 f4];
if sum(R)==0    %% sıfıra bölme hatasını engellemek için
   kont=(sum(R.*f));
else
   kont=(sum(R.*f))/sum(R);
end
```

Benzetimi gerçekleştiren asıl program

```
clc; clear all; close all;
tic
p=60;%parcacik sayısı
P=randn(p,48);
pbest=P;
Vp=zeros(p,48); %parçacik hızı
%% yr set değer dizisi oluşturuluyor
k=2:1:250;
r(1)=0;
r(k)=0.5*sin(2*pi*k/45)+0.2*sin(2*pi*k/15)+0.2*sin(2*pi*k/90);
yr(1)=0; yr(2)=0;
for k=2:1:249
   yr(k+1)=(0.6*yr(k)+0.2*yr(k-1)+r(k));
end
t=1:1:250;
itmax=100;
wmin=0.8; wmax=1.2;
J=[]; EskiJ=[]; cost=[];
```

```
for ite=1:1:itmax
   ite
   w=wmin+((wmax-wmin)/itmax)*(itmax-ite);
   %% parcacıklar sırasıyla sisteme uygulanacak
   for n=1:1:size(P,1)
      %% Her parcacik icin gerekli parametre matrisleri hazırlanıyor
      k=1; i=0;   % (4x4) lük W ağırlık matrisi
      for j=1:1:size(P,2)-32
         m=mod(j,4);
         if m==1
            i=i+1;
            k=1;
         end
         W(i,k)=P(n,j);
         k=k+1;
      end
      cx=P(n,17:1:24); % UF merkez
      sx=P(n,25:1:32); % UF varyans veya eğim
      ax=P(n,33:1:48); % cikis parametreleri
      %% her bir parçacık sisteme uygulanıyor ve J(cost) hesaplanacak
      FLCu=[]; hata=[]; H=zeros(250,4);
      for i=1:1:size(t,2)
         if i==1
            yp(i)=0; yp(i+1)=0;
            FLCu(i)=0;
         elseif i==2
            yp(i+1)=((yp(i)*yp(i-1)*(yp(i)+2.5))/(1+yp(i)^2+yp(i-1)^2))+FLCu(i);
         elseif i==3
            yp(i+1)=((yp(i)*yp(i-1)*(yp(i)+2.5))/(1+yp(i)^2+yp(i-1)^2))+FLCu(i);
         else
            yp(i+1)=((yp(i)*yp(i-1)*(yp(i)+2.5))/(1+yp(i)^2+yp(i-1)^2))+FLCu(i);
         end
         yr1=yr(i);
         yp1=yp(i);
         Ex1=yr1-yp1;  %% hata
         h=H(i,:);
         [kont hy]=TRFNfuzzy(yr1,yp1,h,W,cx,sx,ax);  %% FLC kont sinyali üretiyor
         H(i+1,:)=hy;
         FLCu(i)=kont;
         FLCu=[FLCu; FLCu(i)];
         hata=[hata; Ex1];
      end
      J(n,1)=sqrt((sum(hata.^2)/size(hata,1)));
      if ite==1
         EskiJ(n,1)=J(n,1);
      end
      if J(n,1)<EskiJ(n,1)
         EskiJ(n,1)=J(n,1);
         pbest(n,:)=P(n,:);
      end
   end
   %% tüm parcacıklar için güncellemeler yapılacak
   [P pbest Vp EskiJ]=parcacik_sirala(P,pbest,Vp,EskiJ);
   [P pbest Vp EskiJ]=klasikGA3(P,pbest,Vp,EskiJ);
   [P pbest gbest Vp]=klasikPSO(w,P,pbest,Vp);
   costt=EskiJ(1)
   cost=[cost; costt];
end
%% gbest particle için sistem tekrar çözülüyor(TEST)
```

```
gbest=pbest(1,:);
k=1; i=0;   % (4x4) lük W matrisi
for j=1:1:size(P,2)-32
   m=mod(j,4);
   if m==1
      i=i+1;
      k=1;
   end
   W(i,k)=gbest(1,j);
   k=k+1;
end
cx=gbest(17:1:24);
sx=gbest(25:1:32);
ax=gbest(33:1:48);
FLCu=[]; hata=[]; H=zeros(250,4);
for i=1:1:size(t,2)
   if i==1
      yp(i)=0; yp(i+1)=0;
      FLCu(i)=0;
   elseif i==2
      yp(i+1)=((yp(i)*yp(i-1)*(yp(i)+2.5))/(1+yp(i)^2+yp(i-1)^2))+FLCu(i);
   elseif i==3
      yp(i+1)=((yp(i)*yp(i-1)*(yp(i)+2.5))/(1+yp(i)^2+yp(i-1)^2))+FLCu(i);
   else
      yp(i+1)=((yp(i)*yp(i-1)*(yp(i)+2.5))/(1+yp(i)^2+yp(i-1)^2))+FLCu(i);
   end
   yr1=yr(i);
   yp1=yp(i);
   Ex1=yr1-yp1;  %% hata
   h=H(i,:);
   [kont hy]=TRFNfuzzy(yr1,yp1,h,W,cx,sx,ax);  %% FLC kont sinyali üretiyor
   H(i+1,:)=hy;
   FLCu(i)=kont;
   FLCu=[FLCu; FLCu(i)];
   hata=[hata; Ex1];
end
figure; plot(cost);
figure; plot(yp,'b'); hold on; plot(yr,'r.'); title ('bulunan en iyi durum');
legend('yp','yr')
figure; plot(FLCu,'k'); title ('en iyi durumda uygulanan kontrol sinyali');
toc
```

EK-4A

ANFIS yapısındaki ağ için benzetimi gerçekleştiren program (Gerekli gauss, parçacık sıralama, klasik GA3 ve klasik PSO fonksiyonları Ek3'de verilmiştir. DC motor modeli, Runge-Kutta çözümü ve Sugeno ANFISfuzzy fonksiyonları aşağıda verilmiştir!!!)

```
clc; clear all; close all; tic
p=60;%parcacik sayısı
%% başlangıç değerleri atanıyor
P=randn(p,20); pbest=P;
Vp=zeros(p,4*3+8); %parçacik hızı
itmax=100; wmin=0.8; wmax=1.2; J=[]; EskiJ=[]; cost=[];
for ite=1:1:itmax
   ite
   w=wmin+((wmax-wmin)/itmax)*(itmax-ite);
   %% parcacıklar sırasıyla sisteme uygulanacak
   for n=1:1:size(P,1)
      %% Her parcacik icin kural parametre matrisi(15x3) hazırlanıyor
      k=1; i=0;
      for j=1:1:size(P,2)-8
         m=mod(j,3);
         if m==1
            i=i+1; k=1;
         end
         Qp(i,k)=P(n,j); k=k+1;
      end
      cx1=P(n,13:1:14);
      cx2=P(n,15:1:16);
      sx1=P(n,17:1:18);
      sx2=P(n,19:1:20);
      %% her bir parçacık sisteme uygulanıyor ve J(cost) hesaplanacak
      t=0:0.1:5;
      XDurum=[]; FLCu=[]; hata=[];
      for i=1:1:size(t,2)
         if i==1
            EskiEx1=0; EskiEx2=0;
            tbas=0; tbitis=0; u=0; X1=1; X2=0; %başlangıç koşulları
            Durum=[0 0];
            XDurum=[XDurum; Durum];
         elseif i==2
            tbas=t(i-1);tbitis=t(i); X1=1; X2=0; u=kont;
            [T,Durum]=sec_motor(tbas,tbitis,X1,X2,0,u);
            XDurum=[XDurum; Durum];
         else
            tbas=t(i-1);tbitis=t(i); X1=Durum(1); X2=Durum(2); u=kont;
            [T,Durum]=sec_motor(tbas,tbitis,X1,X2,0,u);
            XDurum=[XDurum; Durum];
         end
         setx1(i)=cos(.5*pi*t(i));
         Ex1=setx1(i)-Durum(:,1);  %% hata ve hatanın değişimi
         yr1=setx1(i);
```

```
            yp1=Durum(:,1);
            kont=sec_fuzzyall2e2(yr1,yp1,Qp,cx1,cx2,sx1,sx2);
            FLCu=[FLCu; kont];
            hata=[hata; Ex1];
            EskiEx1=Ex1;
        end
        J(n,1)=sqrt((sum(hata.^2)/size(hata,1)));
        if ite==1
            EskiJ(n,1)=J(n,1);
        end
        if J(n,1)<EskiJ(n,1)
            EskiJ(n,1)=J(n,1);
            pbest(n,:)=P(n,:);
        end
    end
    %% tüm parcacıklar için güncellemeler yapılacak
    [P pbest Vp EskiJ]=parcacik_sirala(P,pbest,Vp,EskiJ);
    [P pbest Vp EskiJ]=klasikGA3(P,pbest,Vp,EskiJ);
    [P pbest gbest Vp]=klasikPSO(w,P,pbest,Vp);
    costt=EskiJ(1)
    cost=[cost; costt];
end
%% gbest particle için sistem tekrar çözülüyor(TEST)
gbest=pbest(1,:);
k=1; i=0;
for j=1:1:size(P,2)-8
    m=mod(j,3);
    if m==1
        i=i+1; k=1;
    end
    Qp(i,k)=gbest(1,j);
    k=k+1;
    end
cx1=gbest(13:1:14);
cx2=gbest(15:1:16);
sx1=gbest(17:1:18);
sx2=gbest(19:1:20);
XDurum=[]; FLCu=[]; hata=[];
for i=1:1:size(t,2)
    if i==1
        EskiEx1=0; EskiEx2=0;
        tbas=0; tbitis=0; u=0; X1=1; X2=0; %başlangıç koşulları
        Durum=[0 0];
        XDurum=[XDurum; Durum];
    elseif i==2
        tbas=t(i-1);tbitis=t(i); X1=1; X2=0; u=kont;
        [T,Durum]=sec_motor(tbas,tbitis,X1,X2,0,u);
        XDurum=[XDurum; Durum];
    else
        tbas=t(i-1);tbitis=t(i); X1=Durum(1); X2=Durum(2); u=kont;
        [T,Durum]=sec_motor(tbas,tbitis,X1,X2,0,u);
        XDurum=[XDurum; Durum];
    end
    setx1(i)=cos(.5*pi*t(i));
    Ex1=setx1(i)-Durum(:,1);  %% hata ve hatanın değişimi
    yr1=setx1(i);
    yp1=Durum(:,1);
    kont=sec_fuzzyall2e2(yr1,yp1,Qp,cx1,cx2,sx1,sx2);
    FLCu=[FLCu; kont];
```

```
   hata=[hata; Ex1];
   EskiEx1=Ex1;
end
figure; plot(cost); title ('costun degisimi');
figure; plot(t,XDurum(:,1),'b'); hold on; plot(t,setx1,'r.'); title ('bulunan en iyi durum');
figure; plot(t,FLCu,'k'); title ('en iyi durumda uygulanan kontrol sinyali');
toc
```

DC motor modeli:

```
function [zaman,x1x2]=sec_motor(zamanbas,zamanbitis,x1bas,x2bas,d,ubas)
t=zamanbas:0.01:zamanbitis;
U=[]; Y=[]; h=0.01; u=ubas;
for i=1:size(t,2)
   if i==1
      x1=x1bas; x2=x2bas; tilk=0; tson=t(2);
      [x1_son x2_son]=sec_rungekutta(tilk,tson,h,x1,x2,d,u);
      x1=x1_son; x2=x2_son;
      Y=[Y; x1 x2];
   elseif i==size(t,2)
         Y=[Y; x1 x2];
   else
      tilk=t(i); tson=t(i+1);
      [x1_son x2_son]=sec_rungekutta(tilk,tson,h,x1,x2,d,u);
      x1=x1_son; x2=x2_son;
      Y=[Y; x1 x2];
   end
end
zaman=t(:,size(t,2)); x1x2=Y(size(t,2),:);
```

DC motor durum denklemlerini Runge-Kutta metoduyla çözen fonksiyonu:

```
function [x1_son x2_son]=sec_rungekutta(tilk,tson,h,x1,x2,d,u)
%% motor parametreleri
J=0.01; b=0.1; K=0.01; R=1; L=0.5;
a=round((tson-tilk)/h);
for n=1:1:a
   f1= (-b/J)*x1+(K/J)*x2;
   f2= (-K/L)*x1+(-R/L)*x2+(1/L)*u;
   k1x1=h*f1;
   k1x2=h*f2;
   f1=(-b/J)*(x1+0.5*k1x1)+(K/J)*(x2+0.5*k1x2);
   f2=(-K/L)*(x1+0.5*k1x1)+(-R/L)*(x2+0.5*k1x2)+(1/L)*u;
   k2x1=h*f1;
   k2x2=h*f2;
   f1=(-b/J)*(x1+0.5*k2x1)+(K/J)*(x2+0.5*k2x2);
   f2=(-K/L)*(x1+0.5*k2x1)+(-R/L)*(x2+0.5*k2x2)+(1/L)*u;
   k3x1=h*f1;
   k3x2=h*f2;
   f1=(-b/J)*(x1+k3x1)+(K/J)*(x2+k3x2);
   f2=(-K/L)*(x1+k3x1)+(-R/L)*(x2+k3x2)+(1/L)*u;
   k4x1=h*f1;
   k4x2=h*f2;
   yx1=x1+(1/6)*(k1x1+2*k2x1+2*k3x1+k4x1);
   yx2=x2+(1/6)*(k1x2+2*k2x2+2*k3x2+k4x2);
   x1=yx1; x2=yx2;
end
x1_son=yx1; x2_son=yx2;
```

Sugeno ANFIS yapısındaki bulanık-nöral ağ işleyişini gerçekleştiren fonksiyon:

```
function [u]=sec_fuzzyall2e2(yr1,yp1,Qp,cx1,cx2,sx1,sx2)
```

```
%% yr1: set deger
%% yp1: sistemin su andaki durumu
%% Qp:sugeno kural katsayı matrisi
%% yr1 ÜF tanımları ve UF vektörü
Nyr1=gauss(yr1,cx1(1),sx1(1));
Pyr1=gauss(yr1,cx1(2),sx1(2));
yr_UF=[Nyr1 Pyr1];
%% yp1 ÜF tanımları ve UF vektörü
Nyp1=gauss(yp1,cx2(1),sx2(1));
Pyp1=gauss(yp1,cx2(2),sx2(2));
yp_UF=[Nyp1 Pyp1];
%% Kural aktiflikleri (4 kural) Tnorm=ceb.carpım
W=[];
for k=1:1:size(yr_UF,2)
    for l=1:1:size(yp_UF,2)
       wa=yr_UF(k)*yp_UF(l);
       W=[W wa];
    end
end
%% Kuralların işletilmesi // 1.Derece Sugeno kural parametreleri (4x3)P=[p q r]
Kur_sonuc=Qp*[yr1 yp1 1]';
u=W*Kur_sonuc; %% agirlikli toplam
```

EK-4B

TRFN yapısındaki ağ için benzetimi gerçekleştiren asıl program (DC motor modeli ve Runge-Kutta çözüm fonksiyonları yukarıda verilenlerle aynıdır. Gerekli gauss, sigma1, parçacık sıralama, klasik GA3, klasik PSO ve TRFNfuzzy fonksiyonları Ek3'de verilmiştir!!!)

```
clc; clear all; close all; tic
p=60;%parcacik sayısı
%% başlangıç değerleri atanıyor
P=randn(p,48); pbest=P;
Vp=zeros(p,48); %parçacik hızı
itmax=100; wmin=0.8; wmax=1.2; J=[]; EskiJ=[]; cost=[];
for ite=1:1:itmax
   ite
   w=wmin+((wmax-wmin)/itmax)*(itmax-ite);
   %% parcacıklar sırasıyla sisteme uygulanacak
   for n=1:1:size(P,1)
      %% Her parcacik icin gerekli parametre matrisleri hazırlanıyor
      k=1; i=0;   % (4x4) lük W ağırlık matrisi
      for j=1:1:size(P,2)-32
         m=mod(j,4);
         if m==1
            i=i+1; k=1;
         end
         W(i,k)=P(n,j);
         k=k+1;
      end
      cx=P(n,17:1:24); % UF merkez
      sx=P(n,25:1:32); % UF varyans veya eğim
      ax=P(n,33:1:48); % cikis parametreleri
      %% her bir parçacık sisteme uygulanıyor ve J(cost) hesaplanacak
      t=0:0.1:5;
      XDurum=[]; FLCu=[]; hata=[]; H=zeros(51,4);
      for i=1:1:size(t,2)
         if i==1
            EskiEx1=0; EskiEx2=0;
            tbas=0; tbitis=0; u=0; X1=1; X2=0; %başlangıç koşulları
            Durum=[0 0];
            XDurum=[XDurum; Durum];
         elseif i==2
            tbas=t(i-1);tbitis=t(i); X1=1; X2=0; u=kont;
            [T,Durum]=sec_motor(tbas,tbitis,X1,X2,0,u);
            XDurum=[XDurum; Durum];
         else
            tbas=t(i-1);tbitis=t(i); X1=Durum(1); X2=Durum(2); u=kont;
            [T,Durum]=sec_motor(tbas,tbitis,X1,X2,0,u);
            XDurum=[XDurum; Durum];
         end
         setx1(i)=cos(.5*pi*t(i));
         Ex1=setx1(i)-Durum(:,1);  %% hata ve hatanın değişimi
         DEx1=EskiEx1-Ex1;
         yr1=setx1(i);
```

```
            yp1=Durum(:,1);
            h=H(i,:);
            [kont hy]=TRFNfuzzy(yr1,yp1,h,W,cx,sx,ax);  %% FLC kont sinyali üretiyor
            H(i+1,:)=hy;
            FLCu=[FLCu; kont];
            hata=[hata; Ex1];
            EskiEx1=Ex1;
        end
        J(n,1)=sqrt((sum(hata.^2)/size(hata,1)));
        if ite==1
            EskiJ(n,1)=J(n,1);
        end
        if J(n,1)<EskiJ(n,1)
            EskiJ(n,1)=J(n,1);
            pbest(n,:)=P(n,:);
        end
    end
    %% tüm parcacıklar için güncellemeler yapılacak
    [P pbest Vp EskiJ]=parcacik_sirala(P,pbest,Vp,EskiJ);
    [P pbest Vp EskiJ]=klasikGA3(P,pbest,Vp,EskiJ);
    [P pbest gbest Vp]=klasikPSO(w,P,pbest,Vp);
    costt=EskiJ(1)
    cost=[cost; costt];
end
%% gbest particle için sistem tekrar çözülüyor(TEST)
gbest=pbest(1,:);
k=1; i=0;   % (4x4) lük W ağırlık matrisi
for j=1:1:size(P,2)-32
    m=mod(j,4);
    if m==1
      i=i+1; k=1;
    end
    W(i,k)=gbest(1,j);
    k=k+1;
end
cx=gbest(17:1:24); % UF merkez
sx=gbest(25:1:32); % UF varyans veya eğim
ax=gbest(33:1:48); % cikis parametreleri
XDurum=[]; FLCu=[]; hata=[]; H=zeros(51,4);
for i=1:1:size(t,2)
    if i==1
        EskiEx1=0; EskiEx2=0;
        tbas=0; tbitis=0; u=0; X1=1; X2=0; %başlangıç koşulları
        Durum=[0 0];
        XDurum=[XDurum; Durum];
    elseif i==2
        tbas=t(i-1);tbitis=t(i); X1=1; X2=0; u=kont;
        [T,Durum]=sec_motor(tbas,tbitis,X1,X2,0,u);
        XDurum=[XDurum; Durum];
    else
        tbas=t(i-1);tbitis=t(i); X1=Durum(1); X2=Durum(2); u=kont;
        [T,Durum]=sec_motor(tbas,tbitis,X1,X2,0,u);
        XDurum=[XDurum; Durum];
    end
    setx1(i)=cos(.5*pi*t(i));
    Ex1=setx1(i)-Durum(:,1);  %% hata ve hatanın değişimi
    DEx1=EskiEx1-Ex1;
    yr1=setx1(i);
    yp1=Durum(:,1);
```

```
    h=H(i,:);
    [kont hy]=TRFNfuzzy(yr1,yp1,h,W,cx,sx,ax);  %% FLC kont sinyali üretiyor
    H(i+1,:)=hy;
    FLCu=[FLCu; kont];
    hata=[hata; Ex1];
    EskiEx1=Ex1;
end
figure; plot(cost);
figure; plot(t,XDurum(:,1),'b'); hold on; plot(t,setx1,'r.'); title ('bulunan en iyi durum');
figure; plot(t,FLCu,'k'); title ('en iyi durumda uygulanan kontrol sinyali');
toc
```

EK-4C

RFNN yapısındaki ağ için benzetimi gerçekleştiren asıl program (DC motor modeli ve Runge-Kutta çözüm fonksiyonları yukarıda verilenlerle aynıdır. RFNNfuzzy fonksiyonu aşağıda verilmiştir. Gerekli gauss, parçacık sıralama, klasik GA3 ve klasik PSO fonksiyonları Ek3'de verilmiştir!!!)

```
clc; clear all; close all;
tic
p=60;%parcacik sayısı
%% başlangıç değerleri atanıyor
P=randn(p,16);
pbest=P;
Vp=zeros(p,16); %parçacik hızı
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
itmax=100; wmin=0.8; wmax=1.2; J=[]; EskiJ=[]; cost=[];
for ite=1:1:itmax
   ite
   w=wmin+((wmax-wmin)/itmax)*(itmax-ite);
   %% parcacıklar sırasıyla sisteme uygulanacak
   for n=1:1:size(P,1)
      %% Her parcacik icin gerekli parametre matrisleri hazırlanıyor
      fx=P(n,1:1:4); % feedback baglantı agırlıgı
      cx=P(n,5:1:8); % UF merkez
      sx=P(n,9:1:12); % UF varyans
      W=P(n,13:1:16); % cikis  baglantı agırlık
      %% her bir parçacık sisteme uygulanıyor ve J(cost) hesaplanacak
      t=0:0.1:5;
      XDurum=[]; FLCu=[]; hata=[]; F=zeros(51,4);
      for i=1:1:size(t,2)
         if i==1
            EskiEx1=0; EskiEx2=0;
            tbas=0; tbitis=0; u=0; X1=1; X2=0; %başlangıç koşulları
            Durum=[0 0];
            XDurum=[XDurum; Durum];
         elseif i==2
            tbas=t(i-1);tbitis=t(i); X1=1; X2=0; u=kont;
            [T,Durum]=sec_motor(tbas,tbitis,X1,X2,0,u);
             XDurum=[XDurum; Durum];
         else
            tbas=t(i-1);tbitis=t(i); X1=Durum(1); X2=Durum(2); u=kont;
            [T,Durum]=sec_motor(tbas,tbitis,X1,X2,0,u);
            XDurum=[XDurum; Durum];
         end
         setx1(i)=cos(.5*pi*t(i));
         Ex1=setx1(i)-Durum(:,1);  %% hata ve hatanın değişimi
         DEx1=EskiEx1-Ex1;
         yr1=setx1(i);
         yp1=Durum(:,1);
         f=F(i,:);
         [kont fy]=RFNNfuzzy(yr1,yp1,f,fx,cx,sx,W);  %% FLC kont sinyali üretiyor
         F(i+1,:)=fy;
         FLCu=[FLCu; kont];
```

```
            hata=[hata; Ex1];
            EskiEx1=Ex1;
        end
        J(n,1)=sqrt((sum(hata.^2)/size(hata,1)));
        if ite==1
            EskiJ(n,1)=J(n,1);
        end
        if J(n,1)<EskiJ(n,1)
            EskiJ(n,1)=J(n,1);
            pbest(n,:)=P(n,:);
        end
    end
    %% tüm parcacıklar için güncellemeler yapılacak
    [P pbest Vp EskiJ]=parcacik_sirala(P,pbest,Vp,EskiJ);
    [P pbest Vp EskiJ]=klasikGA3(P,pbest,Vp,EskiJ);
    [P pbest gbest Vp]=klasikPSO(w,P,pbest,Vp);
    costt=EskiJ(1)
    cost=[cost; costt];
end
%% gbest particle için sistem tekrar çözülüyor(TEST)
gbest=pbest(1,:);
fx=gbest(1:1:4); % feedback baglantı agırlıgı
cx=gbest(5:1:8); % UF merkez
sx=gbest(9:1:12); % UF varyans
W=gbest(13:1:16); % cikis  baglantı agırlık
XDurum=[]; FLCu=[]; hata=[]; F=zeros(51,4);
for i=1:1:size(t,2)
    if i==1
        EskiEx1=0; EskiEx2=0;
        tbas=0; tbitis=0; u=0; X1=1; X2=0; %başlangıç koşulları
        Durum=[0 0];
        XDurum=[XDurum; Durum];
    elseif i==2
        tbas=t(i-1);tbitis=t(i); X1=1; X2=0; u=kont;
        [T,Durum]=sec_motor(tbas,tbitis,X1,X2,0,u);
        XDurum=[XDurum; Durum];
    else
        tbas=t(i-1);tbitis=t(i); X1=Durum(1); X2=Durum(2); u=kont;
        [T,Durum]=sec_motor(tbas,tbitis,X1,X2,0,u);
        XDurum=[XDurum; Durum];
    end
    setx1(i)=cos(.5*pi*t(i));
    Ex1=setx1(i)-Durum(:,1);  %% hata ve hatanın değişimi
    DEx1=EskiEx1-Ex1;
    yr1=setx1(i);
    yp1=Durum(:,1);
    f=F(i,:);
    [kont fy]=RFNNfuzzy(yr1,yp1,f,fx,cx,sx,W);  %% FLC kont sinyali üretiyor
    F(i+1,:)=fy;
    FLCu=[FLCu; kont];
    hata=[hata; Ex1];
    EskiEx1=Ex1;
end
figure; plot(cost);
figure; plot(t,XDurum(:,1),'b'); hold on; plot(t,setx1,'r.'); title ('bulunan en iyi durum');
figure; plot(t,FLCu,'k'); title ('en iyi durumda uygulanan kontrol sinyali');
toc
```

RFNN yapısındaki bulanık-nöral ağ işleyişini gerçekleştiren fonksiyon:

```
function [kont fy]=RFNNfuzzy(yr1,yp1,f,fx,cx,sx,W)
%% yr1: set deger
%% yp1: sistemin su andaki durumu
%% f: sistemin iç dinamikleri feedback
x1=yr1+f(1)*fx(1);
x2=yr1+f(2)*fx(2);
x3=yp1+f(3)*fx(3);
x4=yp1+f(4)*fx(4);
%% yr1 ÜF tanımları ve UF vektörü
Nyr1=gauss(x1,cx(1),sx(1));
Pyr1=gauss(x2,cx(2),sx(2));
yr_UF=[Nyr1 Pyr1];
%% yp1 ÜF tanımları ve UF vektörü
Nyp1=gauss(x3,cx(3),sx(3));
Pyp1=gauss(x4,cx(4),sx(4));
yp_UF=[Nyp1 Pyp1];
fy=[yr_UF yp_UF];
%% Kural aktiflikleri (4 kural) Tnorm=ceb.carpım
r=[];
for k=1:1:size(yr_UF,2)
    for l=1:1:size(yp_UF,2)
        ra=yr_UF(k)*yp_UF(l);
        r=[r ra];
    end
end
kont=sum(r.*W);
```

KİŞİSEL YAYINLARI

1. Tamer S., Karakuzu C., "Parçacık Sürüsü Optimizasyon Algoritması ve Benzetim Örnekleri", ***ELECO'06 Sempozyumu***

2. Tamer S., Karakuzu C., "Karma (PSO-GA) Rasgele Arama Algoritmasıyla Bulanık-Nöral Kontrolör Eğitimi", ***12. Elektrik Elektronik Bilgisayar Biyomedikal Mühendisliği Ulusal Kongresi*** (incelemede)

ÖZGEÇMİŞ

1983 yılında Denizli'de doğdu. İlk ve orta öğrenimini Denizli'de tamamladı. 2001 yılında üniversite giriş sınavında Kocaeli Üniversitesi Mühendislik Fakültesi Elektronik ve Haberleşme Mühendisliği Bölümünü kazanarak lisans eğitimine başladı. 2005 yılında "Telefon hatları üzerinden ev otomasyon sistemi tasarımı" konusunda bitirme tezi hazırlayarak üniversiteden mezun oldu. Aynı yıl Kocaeli Üniversitesi Fen Bilimleri Enstitüsü Elektronik ve Haberleşme Mühendisliği Ana Bilim Dalında yüksek lisans eğitimine başladı. Şu an mezun olma durumundadır.