G. AVCI 2011

T.C.
NİĞDE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK ELEKRONİK MÜHENDİSLİĞİ ANABİLİM DALI

GERÇEK ZAMANLI UYGULAMALAR İÇİN ABC ALGORİTMASININ FPGA ÜZERİNDE GERÇEKLENMESİ

GÖKMEN AVCI

Haziran 2011

YÜKSEK LİSANS TEZİ

NİĞDE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

T.C.
NİĞDE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

GERÇEK ZAMANLI UYGULAMALAR İÇİN ABC ALGORİTMASININ FPGA ÜZERİNDE GERÇEKLENMESİ

GÖKMEN AVCI

Yüksek Lisans Tezi

Danışman

Yrd. Doç. Dr. Fuat KARAKAYA

Haziran 2011

Gökmen AVCI tarafından **Yrd. Doç. Dr. Fuat KARAKAYA** danışmanlığında hazırlanan **"Gerçek zamanlı uygulamalar için ABC algoritmasının FPGA üzerinde gerçeklenmesi"** adlı bu çalışma jürimiz tarafından Niğde Üniversitesi Fen Bilimleri Enstitüsü **Elektrik Elektronik Mühendisliği** Anabilim Dalında Yüksek Lisans tezi olarak kabul edilmiştir.

Başkan : Prof. Dr. Derviş KARABOĞA (Erciyes Üniversitesi)

Üye : Doç. Dr. Halis ALTUN (Mevlana Üniversitesi)

Üye : Yrd. Doç. Dr. Fuat KARAKAYA (Niğde Üniversitesi)

ONAY:

Bu tez, Fen Bilimleri Enstitüsü Yönetim Kurulunca belirlenmiş olan yukarıdaki jüri üyeleri tarafından …./…./20… tarihinde uygun görülmüş ve Enstitü Yönetim Kurulu'nun …./…./20… tarih ve ……sayılı kararıyla kabul edilmiştir.

…./…./20..

Doç. Dr. Nurettin Acır

Enstitu Müdürü

ÖZET

GERÇEK ZAMANLI UYGULAMALAR İÇİN ABC ALGORİTMASININ FPGA ÜZERİNDE GERÇEKLENMESİ

AVCI, Gökmen
Niğde Üniversitesi
Fen Bilimleri Enstitüsü
Elektrik-Elektronik Mühendisliği Anabilim Dalı

Danışman: Yrd. Doç. Dr. Fuat KARAKAYA

Haziran 2011, 60 sayfa

Bu tez çalışmasında Yapay Arı Kolonisi(Artifical Bee Colony, ABC) algoritması ilk kez FPGA üzerinde donanımsal olarak gerçeklenmiştir. ABC algoritması sürü tabanlı sezgisel (metaheuristic) yeni bir algoritmadır. Arıların doğada polen arama davranışlarından adapte edilmiş ve bir çok optimizasyon probleminde başarı ile kullanılmıştır. Bu çalışmada ABC algoritmasının FPGA üzerinde etkin bir şekilde gerçeklenmesini sağlayan donanım mimarisi önerilmiştir. Önerilen mimari sayesinde algoritma FPGA üzerinde daha az yer kaplamakta ve PC tabanlı yazılım versiyonlarına göre daha hızlı çalışmaktadır. Önerilen mimarinin performansının test edilmesi için literatürden yüksek hesaplama gücü ve hız gerektiren fonksiyonlar seçilmiştir. Aynı zamanda ABC algoritması ile YSA eğitimi problemi de FPGA ortamında gerçeklenmiştir. Yapılan çalışmalar sonucunda donanım üzerinde gerçeklenen yapının hem optimizasyon problemlerinde hemde YSA eğitiminde PC tabanlı uygulamalarla aynı doğruluğu gösterirken, bu uygulamlara göre çok daha hızlı olduğu gözlemlenmiştir.

Anahtar sözcükler: ABC algoritması, FPGA, YSA,

SUMMARY

IMPLEMENTATION OF ARTIFICIAL BEE COLONY (ABC) ALGORITHM ON FPGA FOR REAL-TIME APPLICATIONS

AVCI, Gokmen
Nigde University
Graduate School of Natural and Applied Science
Department of Electrical Electronic Engineering

Supervisor: Assistant Professor Dr. Fuat KARAKAYA

June 2011, 60 pages

In this thesis the first hardware implementation of the artificial bee colony (ABC) optimization on Field-Programmable Gate Arrays (FPGAs) is introduced. The artificial bee colony optimization is a new population–based metaheuristic algorithm. It is adopted from the natural foraging behavior of real honey bees and has been employed to find solutions to wide variety of optimization problems. In this paper we propose an efficient hardware architecture which leads to an efficient implementation with a less space requirement on FPGA and reduced runtime over software based solutions. Proposed architecture is tested with functions selected from the litrature. In addition to that Artificial Neural Network (ANN) tarining with ABC algorithm is also implemented on FPGA. The results indicate that the proposed architacture is as accurate as its software counterpart on PC but much faster than its corresponding software implementation in both function optimization and ANN training.

Key words: FPGAs, ABC algorithm, Artificial Neural Networks(ANNs)

TEŞEKKÜR

Bu tezin hazırlanması sürecinde beni yönlendiren, teşvik eden ve gerek teknik bilgi gerekse de doküman konusunda desteğini esirgemeyen değerli danışmanım Sayın Yrd. Doç. Dr. Fuat KARAKAYA; yüksek lisans eğitimim boyunca beraber çalıştığımız arkadaşım Arş. Gör. Murat PEKER'e ve yardımlarını benden hiç esirgemeyen Doç. Dr. Halis ALTUN'a; hayatım boyunca desteğini benden esirgemeyen canım aileme çok teşekkür ederim.

İÇİNDEKİLER

ÇİZELGELER DİZİNİ

ŞEKİLLER DİZİNİ

BÖLÜM I

GİRİŞ

Optimizasyon (en iyileştirme); bir problemde belirli koşullar altında mümkün olan alternatifler içinden en iyisini seçmektir. "En İyileme" anlamına gelen optimizasyon, her zaman için hedeflenen bir sonuçtur. Optimizasyon teknikleri, yapılmakta olan işin en iyi çözümünü ortaya koymak için kullanılır. Bu teknikler kullanılarak ortaya konulan çözüm, Optimal Çözüm olarak adlandırılır. Hedef her zaman için optimum çözümü yakalayabilmektir. Klasik tekniklerle optimum çözüm genelde deneme yanılma yöntemleriyle aranmıştır. Ama genellikle sub-optimal çözümlerde yeterli olablir. Ancak, problemlerin zorlaşması sebebiyle yapay zeka tabanlı optimizasyon teknikleri önerilmiştir. Önerilen bu tekniklerde doğal hayat taklit edilerek problem çözümleri aranmaktadır.

Evrimsel ve sürü zekası tabanlı optimizasyon algoritmaları literatürde oldukça çok çalışılan konuların başında gelmektedir. Bu algoritmalar karmaşık problemlerin çözümünde göstermiş oldukları başarılardan dolayı birçok bilim dalında, günlük mühendislik problemlerinde ve askeri uygulamalarda sıklıkla kullanılmaktadır. Bu ilginin altında yatan en önemli sebeplerden birisi havacılık, tıp, mühendislik vb. alanlarda uğraşılan problemlerin gittikçe karmaşık bir hal alması ve çoğu ayrık yapıdaki bu problemlerin çözümünde evrimsel ve sürü zekası tabanlı optimizasyon algoritmalarının gösterdiği başarıdır.

Yukarıda bahsi geçen algoritmaların ilham kaynağı ise doğada günlük yaşamlarında zorlu şartlara karşı mücadele veren ve bu mücadele esnasında zeki davranışlar sergileyen canlılardır. Bu canlılara en iyi örneklerden birisi ise bal arısıdır. Bal arıları, bir nektar kaynağı bulup kovana döndüğünde dans yaparak diğer arılara nektarın yeri, uzaklığı ve zenginliği gibi bilgileri aktarır. Bu davranışı örnek alarak 'Yapay Arı Kolonisi' (Artificial Bee Colony-ABC) olarak adlandırılan tasarım yöntemi geliştirilmiştir [1]. ABC yöntemi, 5 binden fazla parametreye sahip karmaşık problemler için en iyileştirilmiş çözümler üretebilmektedir. Literatürde, sayısal tasarım için arının davranışından esinlenilerek geliştirilmiş 10'a yakın yöntem mevcuttur ancak ABC yöntemi bunlardan tamamen farklı ve kendi sınıfındaki yöntemlere göre çok daha

basit, hızlı ve kullanımı daha kolaydır. Bu etkin algoritma birçok alanda kullanılmaya başlanmıştır [2-12].

Bu tez kapsamında literatüre yeni girmiş olmasına rağmen tüm dünyada kabul görmüş olan Yapay Arı Kolonisi(Artifical Bee ColonyABC) algoritması FPGA üzerinde gerçeklenmiştir. ABC algoritmasının gerçek zamanlı uygulamalarda daha etkin kullanılması için yapısal olarak paralel işlem yapma yeteneğine sahip FPGA'lara aktarılması büyük önem arz etmektedir. ABC algoritması paralel işlem yapma yeteneğine sahip donanım üzerinde gerçekleştirilmesi sayesinde askeri ve sivil yüksek hesaplama gücü ve hız gerektiren uygulamalarda kolaylıkla kullanılabilecektir. Yine tez kapsamında algoritmanın donanımsal uyarlanması, literatürden seçilen fonksiyonların optimizasyonu için denenmiş ve performansı diğer bazı algoritmalarla karşılaştırılmıştır. Ayrıca ABC algoritması kullanılarak yapay sinir ağlarının FPGA üzerinde, bilgisayarlara göre daha hızlı ve etkin bir şekilde eğitilebileceği gösterilmiştir.

BÖLÜM II

FPGA ÜZERİNDE GERÇEKLEŞTİRİLEN SÜRÜ ZEKASI TABANLI OPTİMİZASYON ALGORİTMALARI

Evrimsel ve sürü zekası tabanlı optimizasyon algoritmaları literatürde oldukça çok çalışılan konuların başında gelmektedir. Bu algoritmalar kompleks problemlerin çözümünde göstermiş oldukları başarılardan dolayı bir çok bilim dalında, günlük mühendislik problemlerinde ve askeri uygulamalarda sıklıkla kullanılmaktadır. Yarı iletken teknolojilerinde yaşanan gelişmelere bağlı olarak bu algoritmaların donanımsal (DSP, FPGA vb.) uyarlaması ise yoğun bir ilgi görmektedir [13-20]. Kompleks problemlerin çözümlerinde kullanılan bu algoritmaların yoğun bilgi akışının olduğu gerçek zamanlı uygulamalarda kullanılması ise donanımsal uyarlamalarla ivme kazanmıştır. İnsansız hava araçlarının (UAVs) beklenmeyen durumlar karşısında navigasyon rotasını gerçek zamanlı olarak yeniden belirlemesi, füzelerde hedef takibi ve rota tayini ve robotik uygulamalarda robot üzerinde mevcut tüm alıcılardan gelen bilgilerin işlenerek karar verilmesi gibi gerçek zamanlı olarak yoğun bilgi akışı gerektiren uygulamalar evrimsel ve sürü zekası tabanlı optimizasyon algoritmalarının donanımsal olarak kullanılmasını zorunlu hale getirmektedir. Literatürde yapılan çalışmalar göstermektedir ki donanım üzerinde çalışan optimizasyon algoritmaları PC üzerinde çalışan yazılımsal emsallerine göre 100 kat daha hızlı çözüm üretebilmektedirler. Ayrıca donanımsal uyarlamaları daha az güç harcamakta ve daha az alan kaplamaktadır. Bu özellikler ise onları mobil uygulamalar için ideal bir konuma getirmektedir. Donanımsal uygulamalarda ise platform olarak FPGA lar ön plana çıkmaktadır. Bunda en büyük etken FPGA'ların paralel programlamaya uygun mimarileridir. Ayrıca FPGA'ların daha az güç tüketmeleri, çok daha zorlu koşullarda (yüksek sıcaklık) çalışabilmeleri diğer bazı etkenlerdir.

Yarı iletken teknolojilerindeki baş döndürücü gelişmeler kullanılan üretim teknolojileri ile en küçük transistor boyutu 35 nm'lere kadar indirmiştir. Aynı zamanda ASIC devreleri ile FPGA'lar arasında uzunca bir süredir var olan 2 nesil teknoloji farkı kapanmıştır. Günümüzde üretilen FPGA'lar ASIC devrelerle aynı teknolojiye (35 nm) sahiptir. Bahsi geçen gelişmeler sayesinde üretilen FPGA'ların kapasiteleri, hızları, içerdikleri blok RAM miktarı, çarpıcılar, PLL ve diğer bazı kritik donanımsal

modüllerde artmıştır. FPGA'ların içerdikleri bu donanımların yanı sıra, bu FPGA'ların kullanıldığı geliştirme boardlarının sahip olduğu donanımlarda artmıştır (SSRAM, SDRAM, FLASH, DDR2, USB, Ethernet, çeşitli algılayıcılar vb.). Bu sayede önceden donanımsal olarak gerçeklenmesi mümkün olamayan birçok uygulamanın donanım üzerinde geliştirilmesi mümkün olmuştur. Bu tip uygulamalara gösterilebilecek en güzel örnek evrimsel veya sürü zekası tabanlı optimizasyon algoritmalarıdır. Evrimsel veya sürü zekası tabanlı optimizasyon tekniklerinin donanımsal uyarlanmasına dair literatürde mevcut çalışmaların bazıları şu şekilde sıralanabilir:

Juang ve arkadaşları [13] hibrit akıllı bir iniş kontrol mekanizması ve bunun donanım (DSP) üzerinde gerçekleştirilmesini önermişlerdir. Boing firmasının yayınlamış olduğu rapora göre uçak kazalarının %67'lik gibi büyük bir kısmı insan hatalarından kaynaklanmaktadır. Bu kazalarında %47'lik bölümü ise uçağın piste yaklaşması anında veya iniş esnasında gerçekleşmektedir. Yukarıda verilen istatistikler göz önüne alındığında insan hatalarının elimine edilmesi veya minimuma indirilmesi kazaların azaltılması açısından elzemdir. Bu kapsamda tasarlanan kontrol ünitesinde yapay sinir ağı ve PSO (Partical Swarm Optimization) algoritması kullanılmıştır. PSO algoritması yapay sinir ağının eğitiminden sonra optimum kontrol parametrelerinin seçiminde kullanılmıştır. Önerilen yöntem uçağın çok zorlu çevresel koşullara dahi kolayca adapte olmasını sağlamıştır. Donamım üzerinde gerçeklenmesi ise gerçek zamanlı uygulamalarda kullanılmasını mümkün kılmıştır.

Farahani ve arkadaşları [14] ise PSO (Partical Swarm Optimization) algoritmasını Altera Stratix 1S10ES FPGA geliştirme seti üzerinde SOPC tabanlı olarak gerçekleştirmişlerdir. Uygulamada uygunluk fonksiyonu C dilinde yazılmış ve geliştirme bordu üzerine yüklenen Soft-Cpu (Nios II) üzerinde çalıştırılmıştır. PSO algoritmasına ait tüm bloklar ise Verilog donanım tanımlama dili kullanılarak oluşturulmuştur. PSO algoritmasına ait donanım blokları, Soft-Cpu ve gerekli hafıza birimleri Altera'nın SOPC Builder programı kullanılarak Avalon veri iletim hattı vasıtası ile entegre edilmişlerdir. Uygulamanın başarımını ölçmek için literatürden benchmark fonksiyonlar seçilmiş (OneMax, Dejong Test setinden f1 ve f2 fonksiyonları) ve bu fonksiyonların minimum noktalarına yakınsama süreleri hem bilgisayarda hem de donanım üzerinde gerçekleştirilen uygulamada hesaplanmıştır.

Verilen sonuçlar göstermektedir ki donanım üzerinde gerçekleştirilen uygulama, yazılım emsaline göre 100 kat daha hızlı yakınsamaktadır.

Evrimsel algoritmalar (Evolutionary Algorithms- EAs) çok kaliteli sonuçlar üretebilmekte ancak diğer klasik optimizasyon tekniklerine (hill climbing, gradient descented vb.) göre daha uzun sürmektedir. Bunun sebebi ise EA'larda simülasyon tabanlı yaklaşımın kullanılmasıdır. Kompleks problemlerin çözümünde hesaplama süresi olarak en uzun süre uygunluk fonksiyonun değerlendirilmesinde kullanılmaktadır. Literatürde EA'ların donamın üzerinde paralel gerçeklenmesi ile ilgili birçok çalışma mevcuttur ancak bu çalışmaların tamamında uygunluk fonksiyonun değerlendirilmesi aşaması yazılım üzerinde gerçekleştirilmektedir. Süre olarak maliyeti en fazla olan kısmın yazılımsal olarak gerçekleştirilmesi nedeni ile uygulamalar her ne kadar PC versiyonlarına göre hızlı olsalar da donanımdan beklenen performansa ulaşamamaktadırlar. Bir başka yaklaşım ise algoritmanın tamamının donanım üzerinde gerçekleştirilmesidir. Drechler ve arkadaşları [15] Genetik Algoritma (GA) nın tamamını (uygunluk fonksiyonu da dahil) yeniden kullanılabilir modüller seviyesinde VHDL kullanarak FPGA'lar üzerinde gerçekleştirmişlerdir. GA'nın modüler seviyede gerçeklenmesi donanımsal uyarlamada karşılaşılan zorlukları minimuma indirgemiş ve uygulamanın bir çok değişik problemin çözümünde kolaylıkla kullanılmasına imkan sağlamıştır. Gene Avcı, Karakaya ve Altun [16] bir plaka yeri tespit probleminin çözümünde Genetik Algoritmayı Yapay Sinir Ağları ile birlikte donanım üzerinde gerçekleştirmişlerdir.

İnsansız Hava Araçlarında (Unmanned Aerial Vehicle- UAV) rota planı yaparken navigasyon otonomisini arttırmak için birçok teknik geliştirilmiştir. Bu uygulamada en zor kısım ise önceden planlanan rotanın UAV görevde iken hava koşulları, düşman uçakları veya zorlu coğrafik koşullar gibi önceden kestirilemeyen nedenlerden ötürü gerçek zamanlı olarak yeniden planlanmasıdır. Bu süreçte kullanılan teknikler içerisinde en yüksek başarımı gösteren Genetik Algoritma (GA)'dır. GA çözüm uzayını büyük bir başarı ile tararken o ana kadarki en iyi çözümü saklayabilmektedir. Ancak bu süreçte GA yinelemeye dayalı yapısından dolayı karmaşık çözüm uzaylarında oldukça yavaş kalmaktadır. Bu sebebe binaen GA tabanlı gerçek zamanlı navigasyon planlama uygulamaları fazlaca irdelenmemiştir. Allaire ve arkadaşları [17] ise FPGA üzerinde paralel bir yapıda gerçekleştirdikleri GA sayesinde ilk kez GA tabanlı gerçek zamanlı

navigasyon planlama uygulamasını Xilinx Virtex-II Pro FPGA üzerinde 100 MHz lik bir saat darbesi kullanılarak başarı ile gerçekleştirmişlerdir. Yapılan denemeler göstermiştir ki donanımsal uygulama PC (2.8 GHz Pentium 4 işlemcili) tabanlı uygulamaya göre mutasyonda 100 kat, seçme ve çaprazlamada 10,000 kat ve popülasyon güncellemede 50,000 kat daha yüksek performans göstermektedir.

Karınca Kolonisi Optimizasyonu (Ant Colony Optimization- ACO) doğada karınca kolonilerinin yiyecek aramak ve bulunan yiyecek kaynağının yerinin diğer koloni üyeleri ile paylaşılması esnasında kullandıkları karmaşık davranış biçimlerinin taklit edilmesiyle Dorigo ve arkadaşları [18] tarafından geliştirilmiş bir optimizasyon tekniğidir. Karınca kolonilerinde yiyecek arayan karıncalar yiyecek kaynağına ulaşana kadar geçtikleri yollara "pheromone" denilen salgıdan bırakırlar diğer karıncalarda bu salgıları algılayarak yiyecek kaynağının yerini kolayca bulurlar. Yiyecek kaynağına ulaşmak için kullanılan en kısa yolda bu salgı daha yoğun bir şekilde bulunması nedeniyle (mesafe arttıkça salgılanan "pheromone" buharlaşmakta ve dolayısıyla yol üzerindeki yoğunluğu azalmaktadır) daha fazla karınca içgüdüsel olarak en kısa yolu tercih eder. Ayrıca en kısa yolu tercih eden karınca sayısı arttıkça onlarında salgıladığı "pheromone" salgısı nedeni ile bu yol üzerindeki "pheromone" yoğunluğu daha da artar. Scheuermann ve arkadaşları [19] ACO algoritmasını ilk kez FPGA üzerinde gerçekleştirmişlerdir. Bu gerçekleştirmede ACO algoritmasını orijinalinin yerine donanımsal uyarlamaya daha elverişli olan P-ACO (Population-based ACO) versiyonunu kullanmışlardır. Bu versiyonda "pheromone" yerine her bir yinelemede elde edilen en iyi sonuçlar dizinler olarak muhafaza edilmekte ve yeni sonuçlar geldikçe muhafaza edilen sonuçlarla karşılaştırılmakta ve daha iyi olanlar dizindeki diğer çözümlerle yer değiştirmektedir. Bu yer değiştirme işlemi "pheromone" ların artırılması veya buharlaşması işlevini yerine getirmektedir. FPGA (Xilinx Virtex II Pro) üzerinde gerçekleştirilen uygulama PC tabanlı uygulamalara göre 12 kat daha hızlı ve PC tabanlı uygulamalara göre maliyetinin daha düşük olduğu gösterilmiştir.

Karınca Kolonisi Optimizasyon (ACO) algoritmasının FPGA üzerinde donanımsal uyarlanmasına bir başka örnek ise [20] bulanık mantık tabanlı bir kontrol ünitesinin gerçek zamanlı eğitiminde donanım üzerinde ACO algoritmasının kullanılmasıdır. Donanımsal olarak gerçekleştirilmesi nedeni ile PC tabanlı uygulamalara göre daha hızlı çalışmakta, daha az güç gerektirmekte ve boyut olarak çok az yer kaplamaktadır.

Bütün bu özellikleri nedeni ile mobil ve gerçek zamanlı uygulamalarda (örneğin mobil robot kontrolü) rahatlıkla kullanılabilmektedir.

Sürü algoritmalarının donanımsal gerçeklemesine en güzel örneklerden biride WNN (Wavelet Neural Network) yapısının gerçek zamanlı olarak PSO (Partical Swarm Optimization) algoritması kullanılarak eğitiminin gerçekleştirilmesidir [21]. Bu referans da waveletler doğrusal olmayan yapılar olduğundan Taylor seri açılımı ve bakma tabloları (LUT) yaklaşım metotları birlikte kullanılarak donanımsal uyarlamaya uygun hale getirilmiştir.

Yukarıda literatür özetinde de vurgulandığı gibi yarı iletken teknolojilerindeki gelişmelere paralel olarak paralel işlem özelliğine sahip evrimsel esnek hesaplama yöntemlerinin FPGA gibi paralel işlem yapmaya elverişli donanımlar üzerinde uyarlanması oldukça popüler bir araştırma alanıdır. Bunun en önemli sebepleri donanımsal platformlardaki gelişmeler ve gerçek zamanlı uygulamalara duyulan ihtiyaçlardır. Bu tez çalışmasında amaç Karaboğa [22] tarafından önerilen ve Karaboğa ve arkadaşları [2-11] tarafından geliştirilerek uygulanan Yapay Arı Kolonisi algoritmasının ilk kez FPGA üzerinde gerçekleştirilmesi ve yapının performansının yazılım emsali ile kıyaslanmasıdır. ABC algoritmasının donanıma uyarlanması ile ilgili bir çalışma henüz literatürde mevcut değildir. Bu tez çalışması literatürdeki bu boşluğu dolduracaktır.

BÖLÜM III

YAPAY ARI KOLONİSİ (ABC)

Sürü zekası (Swarm Intelligence) son zamanlarda literatürde oldukça yoğun olarak çalışılan konulardan birisidir. Bir takım kurallar kapsamında bir arada yaşayan böcek toplulukları içim sürü (swarm) terimi kullanılmaktadır. Sürü zekası ise sürüdeki bireylerin kendi aralarındaki etkileşim veya iletişim (dans, kimyasal salgılar vb.) vasıtası ile ortaya çıkmaktadır [12]. Sürü zekası sayesinde her türlü kaynak verimli bir şekilde değerlendirilmektedir. Böcek sürülerinin en temel özelliklerinde biride kendi kendine organize olabilmedir. Bu sayede bireysel yada bölgesel bazda gerçekleşen etkileşimler ile küresel seviyede cevap oluşmaktadır. Kendi kendine organize olabilme Bonabeau tarafından dört özellikle karakterize edilmiştir [22]:

1. Pozitif geri besleme
2. Negatif geri besleme
3. Dalgalanmalar
4. Çoklu etkileşimler

Tabiattaki en ilginç sürülerden biriside bal arılarıdır. Bal arıları ortaya koydukları sürü zekası sayesinde çevresel değişimlere uyarlanabilir cevaplar verebilmekte ve iş yükünü dinamik bir şekilde dağıtabilmektedir. Bal arıları çok gelişmiş algılayıcılara, konum belirleme sistemine, fonografik hafızaya ve sezgisel kavrama becerisine sahiptir. Bal arıları sahip oldukları bu yetenekler sayesinde yiyeceklerin saklanması, balın üretilmesi ve dağıtılması, iletişim ve yiyecek arama gibi işleri de en uygun şekilde yerine getirebilmektedirler. Sürü olarak gösterdikleri bu kollektif başarı nedeni ile araştırmacılar arıların davranışlarını modelleme yönünde büyük çaba harcamışlardır [12].

Arı kolonilerinde üç çeşit arı bulunmaktadır: kraliçe arı (queen), erkek arı (drone) ve dişi işçi arılar (worker). Arı kolonilerinde farklı işlere mevcut iş gücünün etkin bir şekilde dağıtılması büyük önem arz etmektedir [23]. Arıların iş dağılımında kullandıkları kriterler (yaş, hormonlar, genetiklerinden kaynaklanan bireysel yatkınlıklar gibi) konusunda çelişkiler vardır [24].

Arıların günlük işleri arasında en önemli aktivite yiyecek aramaktır. Bu aktivite esnasında arıların davranışları üzerinde etkili olabilecek dış etkenlerin (koku, waggle danstan gelen konum bilgisi, başka arılarında kaynağa yönlenmiş olmaları gibi) ve iç etkenlerin (hafızada tutulan kaynak yada koku gibi) üzerinde bir çok çalışma yapılmıştır [25-27]. Besin kaynağı aramakla görevli arılar yuvanın dışından polen ve su kaynakları ararlar. Bu süreç besin kaynağı arayıcılarının (forager) kovandan ayrılmasıyla başlar. Buldukları çiçeklerden veya diğer kaynklardan aldıkları besinleri (polen veya şeker) salgıladıkları özel bir enzimle sindirerek bal yapma sürecini başlatırlar. Kovana döndükten sonra bu işlenmiş besini petek hücrelerine boşaltırlar ve bozulmaması için ekstra enzimler eklerler. Besin ve enzimlerle doldurulan petek hücreleri bal mumu ile örtülürler. Ayrıca besini buldukları kaynaklarla ilgili bilgileri dans hareketleri ile diğer arılara iletirler. Özellikle yiyecek kaynakları hakkında haberleşmek için özel bir dans çeşiti geliştirmişlerdir [12].

Yeni besin kaynakları bulma görevi kaşif arılara (scout) aittir. Kaşif ve yiyecek getiren arılar (işci arılar) arasında temel bir farklılık yoktur. İşçi arılardan bazıları mevcut besin kaynaklarını terkederek kaşif arı görevini üstlenirler ve rastgele yiyecek aramaya başlarlar. Kaşif arıların sayısı tüm koloninin % 5-10'nu geçmez. Kaşif arılar buldukları besin kaynağına ilişkin uzaklık ve yön bilgisinin diğer arılara aktarması gerekir._Bu aktarımı yukarıda da belirtildiği gibi gerçekleştirdikleri dansla yaparlar. Kaşif arı bu dansı kovanın değişik bölgelerinde tekrarlar diğer arılar ise antenleri vasıtası ile kaşif arıya dokunarak besin kaynağının tadı ve kokusu ile ilgili bilgi alırlar.

Arıların bilgi paylaşmak için yaptıkları dans genelde kovanın çıkışında arıların uçuşa geçmeden önce bulundukları yere yakın bir yerde yapılmaktadır. Paylaşılan bilgiye göre bu dansın canlılığa değişmektedir. Nektarın tatlılığı ve miktarı, nektarın çıkarılmasının kolaylığı, kovandan olan uzaklığı, nektarın kıvamı, besinin genel durumu, hava koşulları ve günün hangi vaktinin olduğu gibi etmenler dansı etkilemektedir [22-29].

Verimli kaynaklara ilişkin bilgiler dans alanında diğer arılara iletildiğinden, arılar bir kaç dansı izledikten sonra hangi besin kaynağını tercih edeceğine karar verir. Verimli kaynaklara ilişkin çok daha fazla bilgi aktarımı olduğundan bu kaynakların seçilme olasılığı daha fazladır.

Arılardaki ve diğer böçek sürülerindeki zeki davranışlar içeren süreçlerin incelenmesi araştırmacıların yeni optimizasyon teknikleri geliştirmelerine ilham kaynağı olmuştur. Geliştirilen bu tekniklere en güzel örneklerden biriside Karaboğa tarafından arıların yiyecek arama davranışını modelleyerek ortaya konulan Yapay Arı Kolonisi (ABC) algoritmasıdır[1]. Karaboğa tarfından geliştirilen ABC algoritmasının temel aldığı modelin dayandığı bazı kabuller vardır. Bu kabullerden birincisi her bir yiyecek kaynağın nektarının yalnızca bir görevli arı tarafından toplanmasıdır. Bu kabule göre görevli arıların sayısı toplam yiyecek kaynağı sayısına, işçi arıların sayısı ise gözcü arıların sayısına da eşittir. Nektarı tükenmiş bir kaynağın görevli arısı kaşif arıya dönüşmektedir. Yiyecek kaynaklarına ilişkin lokasyonlar optimizasyon problemine ait olası çözümlere ve kaynakların nektar miktarları ise o kaynaklarla ilgili çözümlerin kalitesine (uygunluk) karşılık gelmektedir.

Karaboğa tarafından önerilen ABC algoritması modeline ait süreç adımları aşağıda alt başlıklar altında açılanmıştır.

3.1 Başlangıç Yiyecek Kaynağı Bölgelerinin Üretilmesi

Arama uzayını arıların yiyecek kaynaklarını içeren bölge olarak düşünürsek, algoritma arama uzayındaki çözümlere karşılık gelen rastgele yiyecek kaynağı yerleri üreterek çalışmaya başlamaktadır. Rastgele yer üretme süreci Eşitlik 3.1 deki gibi hedef fonksiyonun parametrelerinin alt ve üst sınırları arasında rastgele değer üreterek gerçeklenir.

$$x_{ij} = lb + rand[0,1)(ub - lb) \tag{3.1}$$

Burada $\boldsymbol{i} = 1...SN$, $j = 1...D$ ve SN yiyecek kaynağı sayısı ve D ise optimize edilecek parametre sayısıdır. lb hedef fonksiyonunun parametrelerinin alt sınırı. ub ise hedef fonksiyonunun parametrelerinin üst sınırıdır. Aynı zamanda başlangıç aşamasında her kaynağın geliştirilememe sayısını ifade eden $failure_i$ (*i.* kaynağın geliştirilememe sayısı) sayaçları da sıfırlanmaktadır.

3.2 Görevli Arıların Yiyecek Kaynağı Bölgelerine Gönderilmesi

Yiyecek kaynaklarının sayısı görevli arıların sayısına eşittir. İşçi arı çalıştığı yiyecek kaynağı komşuluğunda yeni bir yiyecek kaynağı belirler ve bunun kalitesini değerlendirir. Yeni kaynak daha iyi ise bu yeni kaynağı hafızasına alır [12]. Yeni kaynağın mevcut kaynak komşuluğunda belirlenmesinin benzetimi Eşitlik 3.2 ile tanımlanmaktadır:

$$v_{ij} = x_{ij} + \emptyset_{ij}(x_{ij} - x_{kj}) \tag{3.2}$$

x_{ij} ile gösterilen her bir kaynak için bu kaynağın koordinatlarının bir parametresidir (rastgele seçilen parametresi, *j)* ve bu parametre değiştirilerek x_i komşuluğunda v_i kaynağı bulunur. Eşitlik 3.2de *j, [1,D]* aralığında rastgele üretilen bir tamsayıdır. Rastgele seçilen *j* parametresi değiştirilirken, yine rastgele seçilen x_k komşu çözümünün *(k* € {1,2,.. .*SN}) j.* parametresi ile mevcut kaynağın *j.* parametresinin farkları alınıp [-1,1] arasında rastgele değer alan $\emptyset_{ij}$ sayısı ile ağırlıklandırıldıktan sonra mevcut kaynağın *j.* parametresine eklenmektedir.

Eşitlik 3.2'den de görüldüğü gibi x_{ij} ve x_{kj} arasındaki fark azaldıkça x_{ij} parametresindeki değişim miktarı da azalacaktır. Bu sayede lokal en iyi sonuca yaklaştıkca adaptif olarak değişim miktarı da azalacaktır.

Bu işlem sonucunda v_{ij}'nin parametre limitlerini aşması durumunda parametreye ait olan alt veya üst sınır değerlerine ötelenmektedir (Eşitlik 3.3):

$$v_{ij} = \begin{cases} lb\,, & v_{ij} < lb \\ v_{ij}\,, & lb \leq v_{ij} \leq ub \\ ub\,, & v_{ij} \geq ub \end{cases} \tag{3.3}$$

Sınırlar dahilinde üretilen v_i parametre vektörü yeni bir kaynağı temsil etmekte ve bunun kalitesi hesaplanarak bir uygunluk değeri atanmaktadır (Eşitlik 3.4):

$$fitness_i = \begin{cases} \frac{1}{1+f_i} & f_i \geq 0 \\ 1 + abs(f_i) & f_i < 0 \end{cases} \tag{3.4}$$

f_i, v_i kaynağının yani çözümünün maliyetidir. x_i ile v_i arasında nektar miktarlarına yani uygunluk değerlerine göre bir aç gözlü (greedy) seçme işlemi uygulanır. Yeni bulunan v_i çözümünün daha iyi olması durumunda görevli arı hafızasından eski kaynağa ilişkin lokasyon biligisini silerek v_i kaynağının yerini hafızaya alır. Aksi takdirde görevli arı x_i kaynağına gitmeye devam eder ve x_i çözümü geliştiremediği için x_i kaynağı ile ilgili geliştirememe sayacı *(failure)* bir artar, geliştirdiği durumda ise sayaç sıfırlanır.

3.3 Gözcü Arıların Seleksiyonda Kullanacakları Olasılık Değerlerinin Hesaplanması ve Yiyecek Kaynağı Bölgelerinin Seçilmesi

Görevli arıların kaynakların komşuluklarını belirleme işlemi sonrası gözcü arıların seleksiyon işlemi ile yiyecek kaynaklarını seçme işlemi başlar. Bu işlemde olasılıksal seçme işlemi kullanılıp bu yöntem algoritmada nektar miktarlarına karşılık gelen uygunluk değerleri kullanılarak yapılmaktadır. Temel ABC algoritmasında bu seleksiyon işlemi rulet tekerleği kullanılarak yapılmıştır. Tekerlekteki her bir dilimin açısı uygunluk değeri ile orantılıdır. Bu işlem Eşitlik 3.5 de gösterildiği gibidir.

$$pi = \frac{fitness_i}{\sum_{i=1}^{SN} fitness_i} \tag{3.5}$$

Burada $fitness_i$. *i.* kaynağın kalitesini, *SN* görevli arı sayısını göstermektedir. Bu olasılık hesaplama işlemine göre bir kaynağın nektar miktarı arttıkça (uygunluk değeri arttıkça) bu kaynak bölgesini seçecek gözcü arı sayısı da artacaktır.

Olasılık değerlerinin hesaplanmasından sonra bu değerler baz alınarak gerçekleştirilen rulet tekerleğine göre seçme işleminde her bir kaynak için [0,1] aralığında rastgele sayı üretilir ve p_i değeri bu üretilen sayıdan büyükse görevli arılar gibi gözcü arı da Eşitlik 3.2 yi kullanarak bu kaynak bölgesinde yeni bir çözüm üretir. Yeni çözüm değerlendirilir ve kalitesi hesaplanır. Sonra, yeni çözümle eski çözümün uygunluklarının karşılaştırıldığı ve iyi olanın seçildiği aç gözlü seleksiyon işlemine tabi tutulur. Yeni çözüm daha iyi ise eski çözüm yerine bu çözüm alınır ve çözüm geliştirememe sayacı sıfırlanır. Eski çözümün uygunluğu daha iyi ise bu çözüm

muhafaza edilir ve geliştirememe sayacı bir artırılır. Bu süreç tüm gözcü arılar yiyecek kaynağı bölgelerine dağılana karlar devam eder [12].

3.4 Nektarı Tükenmiş Kaynağı Bırakma Kriteri

Bir çevrim sonunda tüm görevli ve gözcü arılar arama süreçlerini tamamladıktan sonra çözüm geliştirememe sayaçları kontrol edilir. Bir arının bir kaynaktan faydalanıp faydalanamadığı, yani gidip geldiği kaynağın nektarının tükenip tükenmediği çözüm geliştirememe sayaçları aracılığıyla bilinir. Bir kaynak için çözüm geliştirememe sayacı belli bir eşik değerinin üzerindeyse, artık bu kaynağın görevli arısının tükenmiş olan o çözümü bırakıp kendisi için başka bir çözüm araması gerekmektedir. Bu da biten kaynakla ilişkili olan görevli arının kaşif arı olması anlamına gelmektedir. Kaşif arı haline geldikten sonra, bu arı için rastgele çözüm arama süreci başlar (Eşitlik 3.1). Kaynağın tükendiğinin belirlenmesi için kullanılan eşik değeri ABC algoritmasının önemli bir kontrol parametresidir ve *"limit"* olarak adlandırılmaktadır. Temel ABC algoritmasında her çevrimde sadece bir kaşif arının çıkmasına izin verilir.

Tüm bu birimler arasındaki ilişki ve döngü Şekil 3.1'deki gibi bir akış diyagramı ile ifade edilebilir.

3.5 Seleksiyon Mekanizmaları

ABC algoritması 4 farklı seleksiyon işlemi kullanmaktadır. Bunlar [12]:

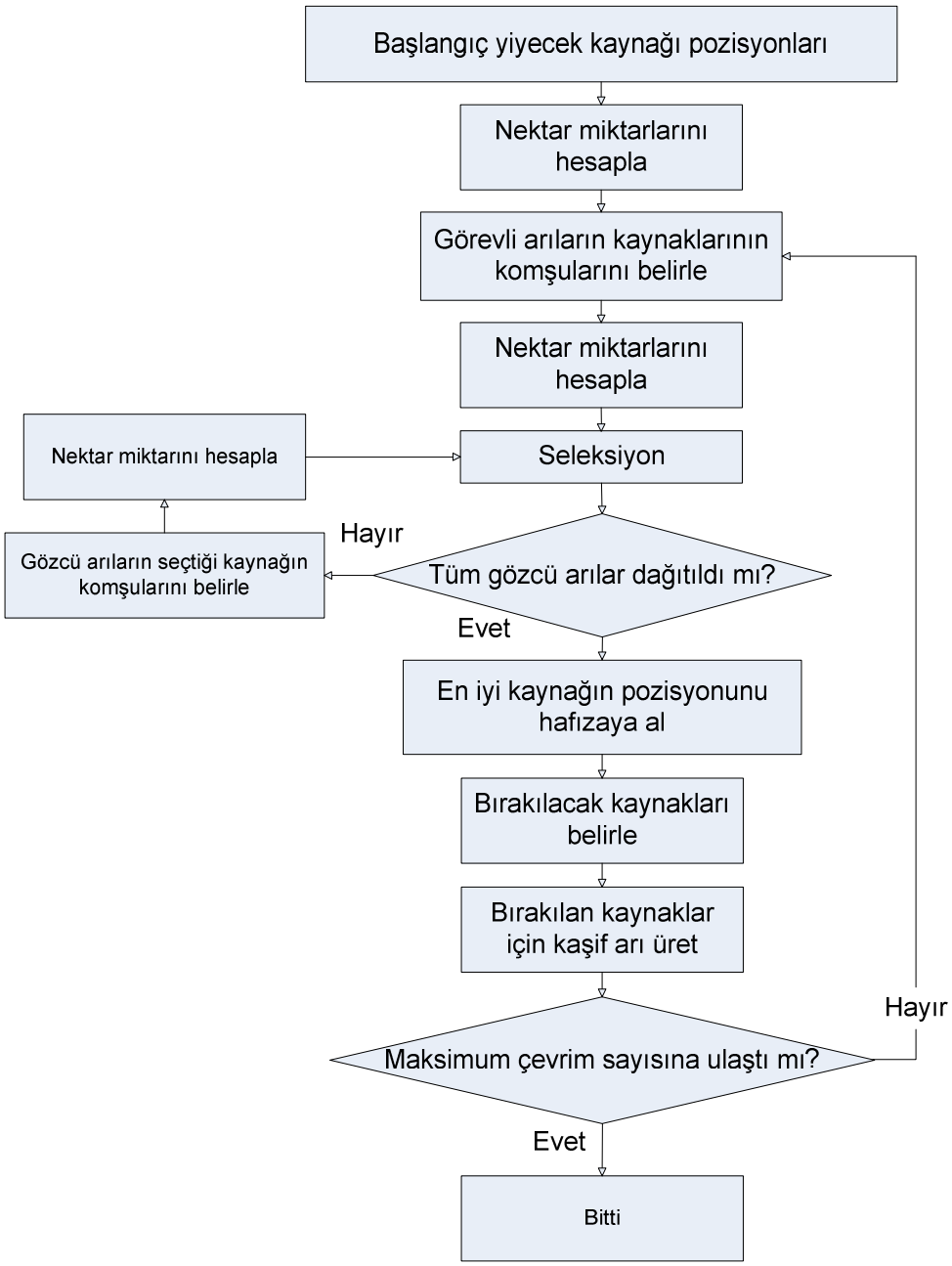
1) Potansiyel iyi kaynakların belirlenmesine yönelik Eşitlik 3.5 ile olasılık değerlerinin hesaplandığı global olasılık temelli seleksiyon süreci

2) Görevli ve gözcü arıların renk, şekil, koku gibi nektar kaynağının türünü belirlemesini sağlayan görsel bilgiyi kullanarak bir bölgede kaynağın bulunmasına vesile olan bölgesel olasılık tabanlı seleksiyon işlemi (Eşitlik 3.2)

3) İşçi ve gözcü arıların daha iyi olan kaynağı belirlemek amacıyla kullandıkları aç gözlü seleksiyon

4) Kaşif arılar tarafından Eşitlik 3.1 aracılığı ile gerçekleştirilen rastgele seleksiyon.

Bütün bu seleksiyon metotlarının bir arada kullanılmasıyla ABC algoritması hem iyi bir global araştırma hem de iyi bir bölgesel araştırma yapabilmektedir [12].



Şekil 3.1. ABC algoritmasının akış diyagramı

3.6 ABC Algoritmasının Adımları

Önceki bölümlerde anlatılan ABC algoritmasının genel hatları aşağıda bir sözde kod ile özetlenmiştir.

1. Eşitlik 3.1 aracılığıyla tüm x_{ij}, $i = 1...$ SN, $j = 1...$ D. çözümlerine başlangıç değerlerinin atanması ve çözüm geliştirememe sayaçlarının sıfırlanması ($failure_i = 0$)
2. $f(x_i)$ fonksiyon değerlerinin ve bu değerlere karşılık gelen uygunluk değerlerinin, $fitness_i$, hesaplanması.
3. Repeat
4. for i=1 to SN loop
5. Eşitlik 3.2'yi kullanarak X_i çözümünün görevli arısı için yeni bir kaynak üret, v_i ve $f(v_i)$'yi (3.5) eşitliğinde yerine koyarak bu çözümün uygunluk değerini hesapla.
6. v_i ve x_i arasında aç gözlü seleksiyon işlemi uygula ve daha iyi olanı seç.
7. x_i çözümü gelişememişse çözüm geliştirememe sayacını bir artır, $failure_i = failure_i + 1$. gelişmişse sıfırla, $failure_i = 0$.
8. end for
9. Eşitlik 3.5 ile gözcü arıların seçim yaparken kullanacakları uygunluk değerine dayalı olasılık değerlerlerini, pi_i hesapla.
10. $t = 0$, $i = 1$
11. repeat
12. if $randam < p_i$ then
13. Eşitlik 3.2'i kullanarak gözcü arı için yeni bir kaynak, v_j üret
14. v_i ve x_i arasında aç gözlü seleksiyon işlemi uygula ve daha iyi olanı seç.
15. x_i *çözümü* gelişememişse çözüm geliştirememe sayacını bir artır, $failure_i = failure_i + 1$, gelişmişse sıfırla, $failure_i = 0$.
16. $t = t + 1$
17. end if
18. until $t = SN$
19. if $max(failure_i) > limit$ then
20. x_{ij} Eşitlik 3.1 ile üretilen rastgele bir çözümle değiştir.
21. end if
22. En iyi çözümü hafızada tut
23. until çevrim sayısı = maksimum çevrim sayısı

BÖLÜM IV

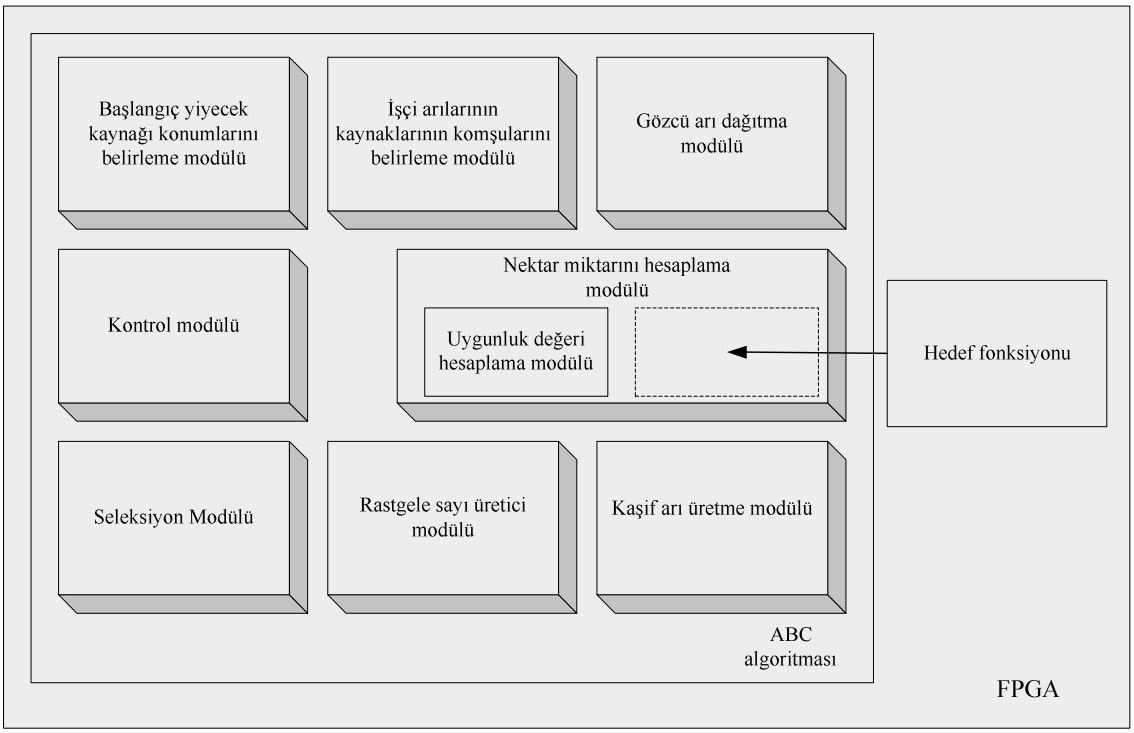
ABC ALGORİTMASININ DONANIMSAL OLARAK GERÇEKLENMESİ

ABC algoritması Şekil 3.1 de verilen akış diyagramında görüldüğü üzere belirli bölümlerden oluşmaktadır ve sıralı bir algoritma olup her bir adım diğer adımı beklemektedir. Ancak donanım üzerinde tasarlanacak olan sistemin sıralı çalışması halinde gerçek zamanlı sistemler için yeterli derecede başarı elde edilemeyecektir. Bu yüzden sıralı çalışan her bir bölüm Şekil 4.1 deki gibi modüllere ayrılarak her modül kendi içlerinde paralel çalışacak şekilde VHDL donanım tanımlama diliyle FPGA üzerinde yeniden tasarlanmıştır. Modüler bir yapıda donanım oluşturmanın birçok faydaları vardır bunlardan en önemlileri şu şekilde sıralanabilir:

(i) Her bir blok tek başına oluşturulduğu için problem kolaylaşır,
(ii) Herhangi bir blokta değişiklik yapıldığında tüm blokların derlenmesi yerine sadece değişiklik yapılan blok derlenir böylece derleme süresi oldukça kısalır,
(iii) Herhangi bir hata oluşması durumunda hatanın izinin sürülmesi oldukça kolay olur,
(iv) Her bir modülün donanım üzerinde kapladığı alan ölçülebilir ve çok alan kaplayan modüller daha detaylı bir çalışma ile optimize edilebilir. Yine aynı şekilde çalışma süresi olarak da her bir modül tek başına değerlendirilebilir ve çok uzun süre gerektiren modüller optimize edilebilir.

4.1 Donanım Üzerinde Oluşturulan Modüller

Donanım modülleri VHDL donanım tanımlama dili kullanılarak kodlanmıştır. Her bir modülle ilgili detaylar aşağıda verilmiştir.



Şekil 4.1 ABC Algoritmasının FPGA üzerindeki modüler yapısı

4.1.1 Başlangıç yiyecek kaynağı konumlarının üretilmesi modülü

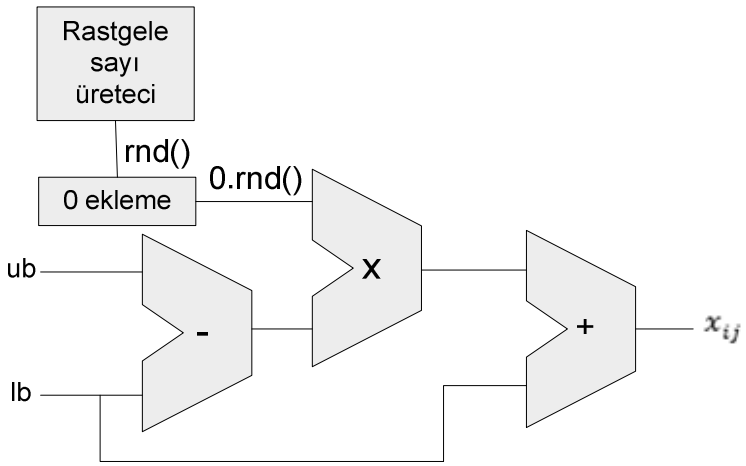
Arama uzayını arıların yiyecek kaynaklarını içeren bölge olarak düşünürsek, algoritma arama uzayındaki çözümlere karşılık gelen rastgele yiyecek kaynağı yerleri üreterek çalışmaya başlamaktadır. Rastgele yer üretme süreci Eşitlik 4.1 deki gibi hedef fonksiyonun parametrelerinin alt ve üst sınırları arasında rastgele değer üreterek gerçeklenir.

$$x_{ij} = lb + rand[0,1)(ub - lb) \tag{4.1}$$

Burada $\boldsymbol{i} = 1...SN$, $j = 1...D$ ve SN yiyecek kaynağı sayısı ve D ise optimize edilecek parametre sayısıdır. lb hedef fonksiyonunun parametrelerinin alt sınırı. ub ise hedef fonksiyonunun parametrelerinin üst sınırıdır. Aynı zamanda başlangıç aşamasında her kaynağın geliştirilememe sayısını ifade eden $failure_i$ (*i.* kaynağın geliştirilememe sayısı) sayaçları da sıfırlanmaktadır.

Bu modülde her bir yiyecek kaynağının her bir parametresinin rastgele sayı üretilmesi için sahte rastgele sayı üreteçleri (Pseudo Random Number Generators, PRNG)'nden

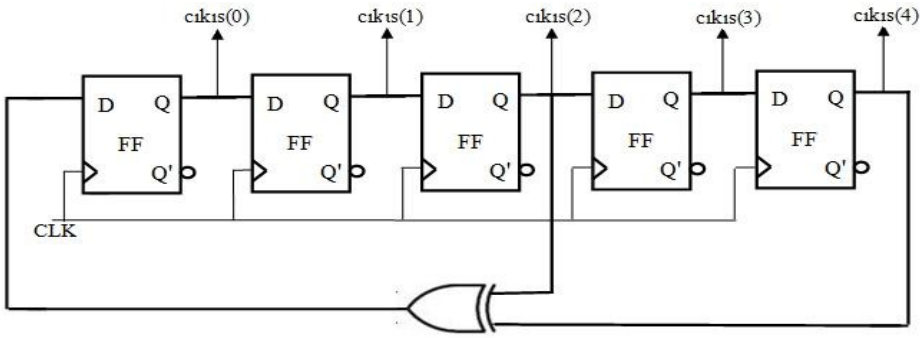
yararlanılmıştır. PRNG'ler donanımsal olarak gerçekleştirilirken LFSR (Linear Feedback Shift Register) olarak bilinen doğrusal geri beslemeli kaydırma yazmaçları kullanılır. Bu nedenle her bir saat darbesinde bir sayı üretilir. Bu modülün donanıma uyarlanmış şekli Şekil 4.2 de görüldüğü gibidir.



Şekil 4.2 Hedef fonksiyonunun parametrelerini belirlenen aralıklarda donanım üzerinde rastgele olarak üretilmesinin gösterilmesi

4.1.2 Rastgele sayı üreteci modülü

Sürü zekasına dayalı optimizasyon tekniklerinin çoğunda olduğu gibi ABC algoritmasında da rastgele popülasyon üretmek büyük bir önem taşımaktadır. ABC algoritmasında başlangıç popülasyonu üretme aşamasında ve yeni kaşif arı üretme aşamasında rastgele birey üretilmesine ihtiyaç vardır. Literatürde en çok kullanılan rastgele sayı üreteçleri gerçek rastgele sayı üreteçleri (True Random Number Generators, TRNG) ve sahte rastgele sayı üreteçleri (Pseudo Random Number Generators, PRNG) olarak sıralanabilir. TRNG'ler çıkışlarını fiziksel bir gürültü kaynağından türetirler. PRNG'ler ise girişlerine verilen ve tohum (seed) olarak adlandırılan girdileri kullanarak görünüşte rastgele olan sayı dizileri üretirler. Her iki rastgele sayı üretecini de FPGA üzerinde gerçekleştirmek mümkündür [Tsoi, 2003]. Şekil 4.3 de PRNG tipi bir rastgele sayı üretecinin devre şeması görülmektedir.



Şekil 4.3 LFSR tabanlı 5 bitlik PRNG

PRNG ler donanımsal olarak gerçekleştirilirken LFSR (Linear Feedback Shift Register) olarak bilinen doğrusal geri beslemeli kaydırma yazmaçları kullanılır. Şekil 4.4'de 5 bitlik basit bir PRNG'nin VHDL kodu verilmiştir. ***N*** bitlik PRNG'nin periyotu 2^N-1 dir. Şekil 4.3 de görüldüğü üzere 5 bitlik bir PRNG elde etmek için kaydırma yazmaçlarının 4. ve 2. bitleri exnor'lanıp yazmaçlara yeni giriş olarak verilmektedir. Farklı bitlerde LFSR tabanlı PRNG ler oluşturmak için Çizelge 4.1 kullanılabilir.

```
process(clk)
        variable seedn : std_logic_vector(5 downto 0) :=(OTHERS=>'0');
        variable a : std_logic :='0';
begin
        if clk'event and clk='1' then
                if a='0' then
                        seedn:='1'&seed_in;
                        a:='1';
                end if;
                seedn(4 downto 0):=seedn(3 downto 0)&(seedn(4) xnor seedn(2));
                cikis<=seedn(4 downto 0);
        end if;
end process;
```

Şekil 4.4 LFSR tabanlı 5 bitlik PRNG'nin VHDL kodu

Çizelgeden de görüldüğü üzere istenildiği büyüklükte PRNG yapmak mümkündür. Çizelgenin ilk sütunundaki sayılar oluşturulacak PRNG'nin toplam bit sayısını ikinci sütunu ise PRNG'nin hangi bitlerinin exnor'lanacagını göstermektedir.

Çizelge 4.1 Farklı bitlerde LFSR tabanlı PRNG'ler için exnor'lanması gereken bitler.

n	xnor	n	xnor	n	xnor
3	3,2	13	13,4,3,1	23	23,18
4	4,3	14	14,5,3,1	24	24,23,22,17
5	5,3	15	15,14	25	25,22
6	6,5	16	16,15,13,4	26	26,6,2,1
7	7,6	17	17,14	27	27,5,2,1
8	8,6,5,4	18	18,11	28	28,25
9	9,5	19	19,6,2,1	29	29,27
10	10,7	20	20,17	30	30,6,4,1
11	11,9	21	21,19	31	31,28
12	12,6,4,1	22	22,21	32	32,22,2,1

4.1.3 Nektar miktarını hesaplama modülü

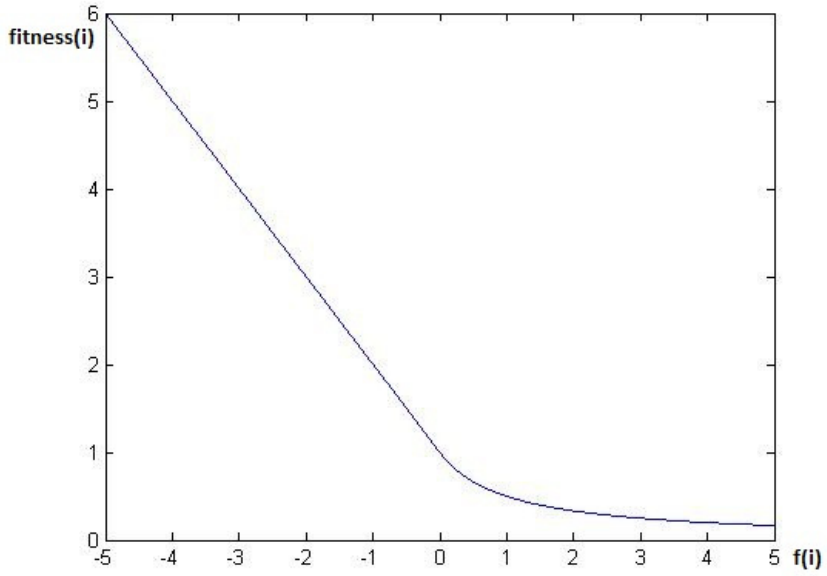
ABC algoritmasının hedefi arıların kovan etrafında en zengin ve kovana en yakın olan yiyecek kaynağının bulunmasıdır. Bu yaklaşımla her bir yiyecek kaynağının bir maliyeti çıkartılır ve algoritma maliyeti en düşük olan bölgeyi bulmaya çalışır. Bu mantıkla çözülmek istenen problemler optimize edilmektedir. Bu problem çok parametreli bir fonksiyonun minimumunun veya maksimumu bulunması olabileceği gibi büyük bir bilgi seti içerisinden optimum bir değerin hesaplanması da olabilir.

Nektar miktarını bir diğer değişle kaynak maliyetini hesaplama modülü Şekil 4.1'de de görüldüğü gibi iki parçadan oluşmaktadır. Bunlar çözülmek istenen hedef problemin fonksiyonu ve hedef fonksiyonun uygunluk fonksiyonu dur. Hedef fonksiyonu çözülecek olan probleme özgü olduğundan Şekil 4.1'de ABC algoritması dışında gösterilmiştir. Çünkü her problem için özel hedef fonksiyonu modülü yazılması gerekmektedir ve oluşturulan ABC yapısında Şekil 4.1 deki gibi dışarıdan eklenerek çözülmesi sağlanmaktadır. Hedef fonksiyona örnek olarak literatürde çokça kullanılan Rosenbrock fonksiyonu (Eşitlik 5.1) gösterilebilir. ABC algoritmasının FPGA üzerine uyarlanmasında performans testi için kullanılan hedef fonksiyonları ayrıca sonraki bölümlerde anlatılacaktır.

Nektar miktarını hesaplama modülünün ikinci parçası ise uygunluk değeri hesaplama modülüdür. ABC algoritmasında genel amaç nektarın en çok olduğu yiyecek alanının belirlenmesidir. Bu açıdan bakıldığında algoritma hedef fonksiyonu maksimum yapan değeri hesaplamaktadır. Ancak yukarıda örnek olarak verilen Rosenbrock fonksiyonunda veya diğer algoritmalarda da olduğu gibi hedef minimumu bulmaksa minimize edilen fonksiyondan gelen cevapların Eşitlik 4.2 kullanılarak minimumların maksimuma çevrilmesi gerekir. Bu modül bu işlevi yerine getirmektedir.

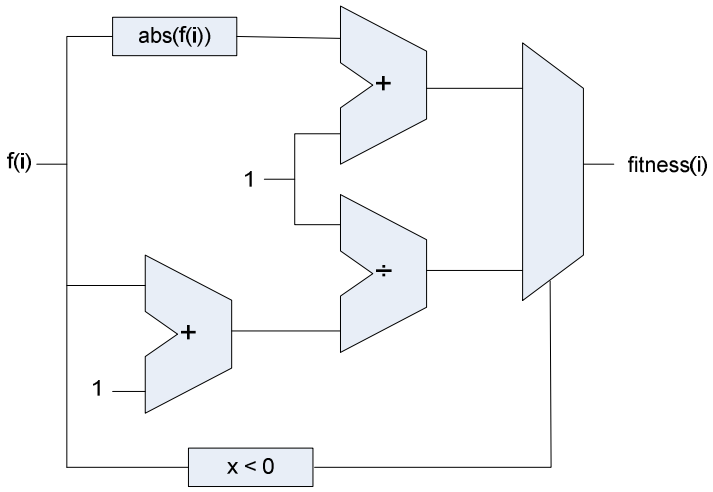
$$fitness(i) = \begin{cases} 1 + abs(f(i)), & x < 0 \\ 1/(f(i) + 1), & x \geq 0 \end{cases} \tag{4.2}$$

Eşitlik 4.2 deki *fitness(i)* değeri ABC algoritması tarafından bulunan aday çözümlerin hedef fonksiyon cevabı bir başka değişle çözümün hatasıdır. *f(i)* yeni bir kaynağı temsil etmekte ve kaynağın kalitesi hesaplanarak bir uygunluk değeri atanmaktadır. Uygunluk değeri hesaplama modülünün hedef fonksiyonuna bağlı olarak değişimi Şekil 4.5 de verilmiştir.



Şekil 4.5 Uygunluk değerlerinin hedef fonksiyonuna göre değişimi

Bu modülün donanıma uyarlanması Şekil 4.6 de görüldüğü gibidir.



Şekil 4.6 Uygunluk değeri hesaplama modülünün donanıma uyarlanmış şekli

4.1.4 Görevli arıların komşu kaynakları belirleme modülü

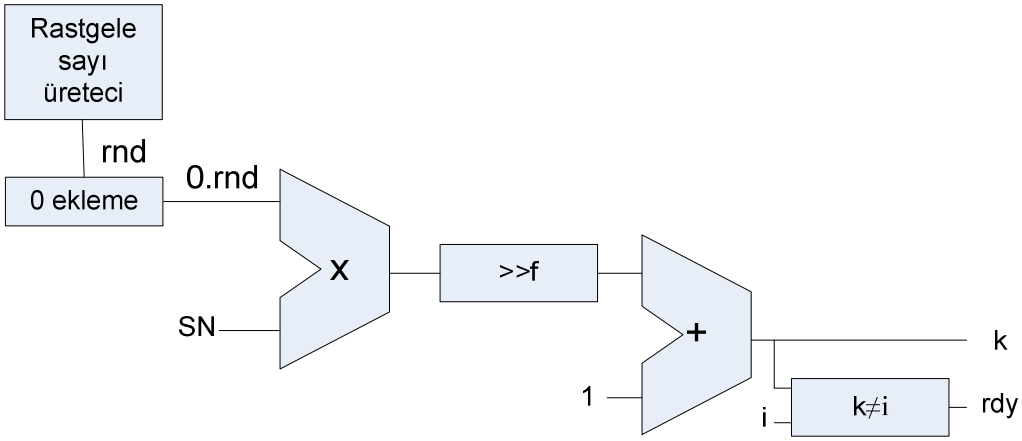
Yiyecek kaynaklarının sayısı görevli arıların sayısına eşittir. İşçi arı çalıştığı yiyecek kaynağı komşuluğunda yeni bir yiyecek kaynağı belirler ve bunun kalitesini değerlendirir. Yeni kaynak daha iyi ise bu yeni kaynağı hafızasına alır [12]. Yeni kaynağın mevcut kaynak komşuluğunda belirlenmesinin donanımsal benzetimi Eşitlik 4.3 ile tanımlanmaktadır:

$$v(i,j) = x(i,j) + [2 * RND(0,1) - 1] * (x(i,j) - x(k,j)) \quad (4.3)$$

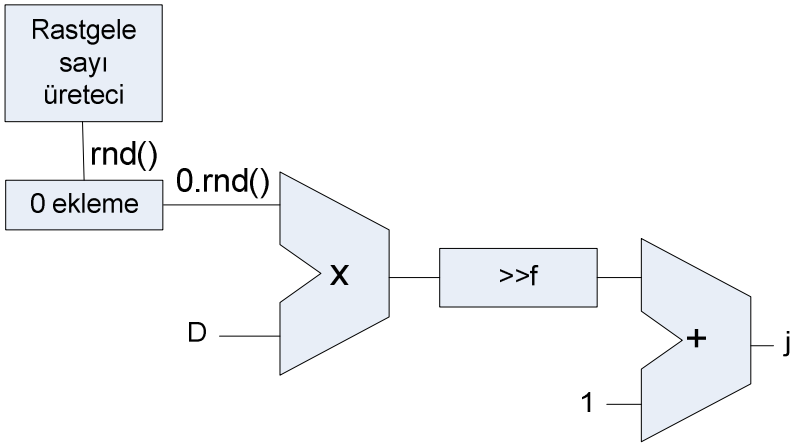
x(i,j) bir kaynak için bu kaynağın yani çözümün tek bir parametresi (rastgele seçilen parametresi, *j)* değiştirilerek *x(k)* komşuluğunda *v(i)* kaynağı bulunur. Eşitlik 4.3'de *j [1,D]* aralığında rastgele üretilen bir tamsayıdır. Rastgele seçilen *j* parametresi değiştirilirken, yine rastgele seçilen x(*k)* komşu çözümünün *(k* € {1,2,.. .*SN}) j.* parametresi ile mevcut kaynağın *j.* parametresinin farkları alınıp [-1,1] arasında rastgele değer alan bir sayı ile ağırlıklandırıldıktan sonra mevcut kaynağın *j.* parametresine eklenmektedir. *j* ve *k* nın değerleri Eşitlik 4.4 ve 4.5 deki gibi hesaplanıp donanımsal gösterimleri Şekil 4.7 ve Şekil 4.8 deki gibidir.

$$k = [0.rnd\ x\ SN] + 1 \quad (4.4)$$

$$j = [0.rnd\ x\ D] + 1 \quad (4.5)$$

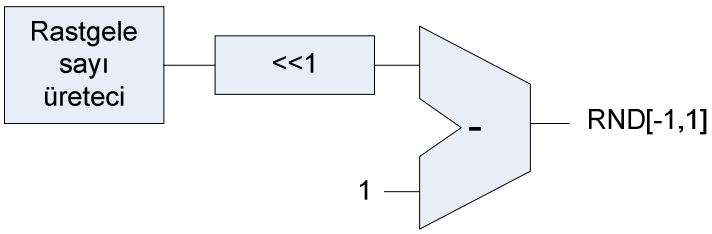


Şekil 4.7 Yiyecek kaynaklarının rastgele seçilme işlemi gösterimi



Şekil 4.8 Hedef fonksiyonu parametrelerinin rastgele seçilme işlemi gösterimi

[-1,1] arasında rastgele değer LFSR yardımıyla elde edilen sayıyı sola doğru kaydırma yöntemi kullanılarak 2 ile çarpılıp 1 çıkarılması ile elde edilmektedir. Bu aralıkta değer üretme yöntemi donanım üzerinde Şekil 4.9 deki gibi üretilmektedir.

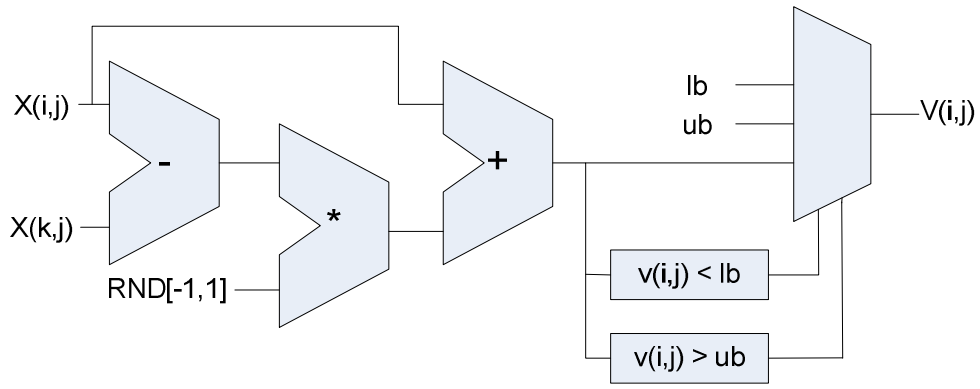


Şekil 4.9 [-1,1] arasında rastgele sayı üretme modülünün gösterimi

Görevli arıların Eşitlik 4.3 deki işlem sonucunda ürettikleri v_{ij}'nin daha önceden belli olan parametre sınırlarını aşması durumunda *j.* parametreye ait olan alt(lb) veya üst(ub) sınır değerlerine Eşitlik 4.6 ya göre ötelenmektedir.

$$v_{ij} = \begin{cases} lb, & v_{ij} < lb \\ v_{ij}, & lb \leq v_{ij} \leq ub \\ ub, & v_{ij} \geq ub \end{cases} \tag{4.6}$$

Sınırlar dahilinde üretilen v_i parametre vektörü yeni bir kaynağı temsil etmekte ve bu yiyecek kaynağının kalitesi Eşitlik 4.2 ile hesaplanarak bir uygunluk değeri atanmaktadır. Şekil 4.10 da görevli arıların çözüm üretme modülünün donanıma uyarlanmış şekli gösterilmektedir.



Şekil 4.10 Görevli arıların kaynakların komşularını belirleme modülü gösterimi

4.1.5 Gözcü arı dağıtım modülü

Modülde nektar miktarlarına karşılık gelen uygunluk değerleri kullanılarak olasılıksal seçme işlemi yapılmaktadır. Temel ABC algoritmasında bu seleksiyon işlemi rulet tekerleği kullanılarak yapılmıştır. Tekerlekteki her bir dilimin açısı uygunluk değeri ile orantılıdır. Bu işlemin Eşitlik 4.7 deki gibidir.

$$p_i = \frac{fitness_i}{\sum_{i=1}^{SN} fitness_i} \tag{4.7}$$

Burada $fitness_i$ *i.* kaynağın kalitesini, *SN* görevli arı sayısını göstermektedir. Bu olasılık hesaplama işleminde $fitness_i$ değeri büyük olan yiyecek kaynağını seçecek gözcü arı sayısı da artacaktır. Algoritmada olasılık değerleri hesaplandıktan sonra bu değerler kullanılarak rulet tekerleğine göre seçim işleminde her bir kaynak için [0.1] aralığında rastgele sayı üretilir ve p_i değeri bu üretilen sayıdan büyükse görevli arılar gibi gözcü arı da Eşitlik 4.3 i kullanarak bu kaynak bölgesinde yeni bir çözüm üretir. Yeni çözüm değerlendirilir ve kalitesi hesaplanır. Bu modülde sadece Eşitlik 4.7 işlemi yapılıp bulunan olasılık değerlerine göre bir yiyecek kaynağı seçilir. Bu modülden sonra Şekil 4.10 daki görevli arı komşuluk modülü çalışır. Bu modüldeki *i* değeri rulet tekeri ile seçilmiş yiyecek kaynağıdır. Modülün çalışma mantığının daha iyi anlaşılması için aşağıda bir sözde kod verilmiştir.

```
While (1 < SN)
{
        if  0.rnd() < p_i
        {
                k ve j değerlerini belirleme modülleri
                görevli isçi arı çözüm üretme modülü
                l++;
        }
        i++;

        if i >SN
        {
                i=1;
        }
}
```

Gözcü arı modülü yiyecek kaynağı sayısı kadar çalıştırılır.

4.1.6 Seleksiyon modül

Bu modülde görevli ve gözcü arılarının üretmiş olduğu yeni çözümlerle eski çözümün uygunluklarının karşılaştırıldığı ve iyi olanın seçildiği aç gözlü seleksiyon işlemine tabi tutulur. Yeni çözüm daha iyi ise eski çözüm yerine bu çözüm alınır ve çözüm geliştirememe sayacı $(failure_i)$ sıfırlanır. Eski çözümün uygunluğu daha iyi ise eski çözüm muhafaza edilir ve geliştirememe sayacı $(failure_i)$ bir artırılır. Bu modüle ait sözde kod aşağıda verilmiştir

```
if (fitness_new(i)>fitness(i))
        {
                failure(i)=0;
                fitness(i) =fitness_new(i);
                f(i)=f_new(i);
                x(i,:)=x_new(i,:);
        }
else
        {
                failure(i)= failure(i)+1;
                fitness(i) =fitness(i);
                f(i)=f(i);
                x(i,:)=x(i,:);
        }
```

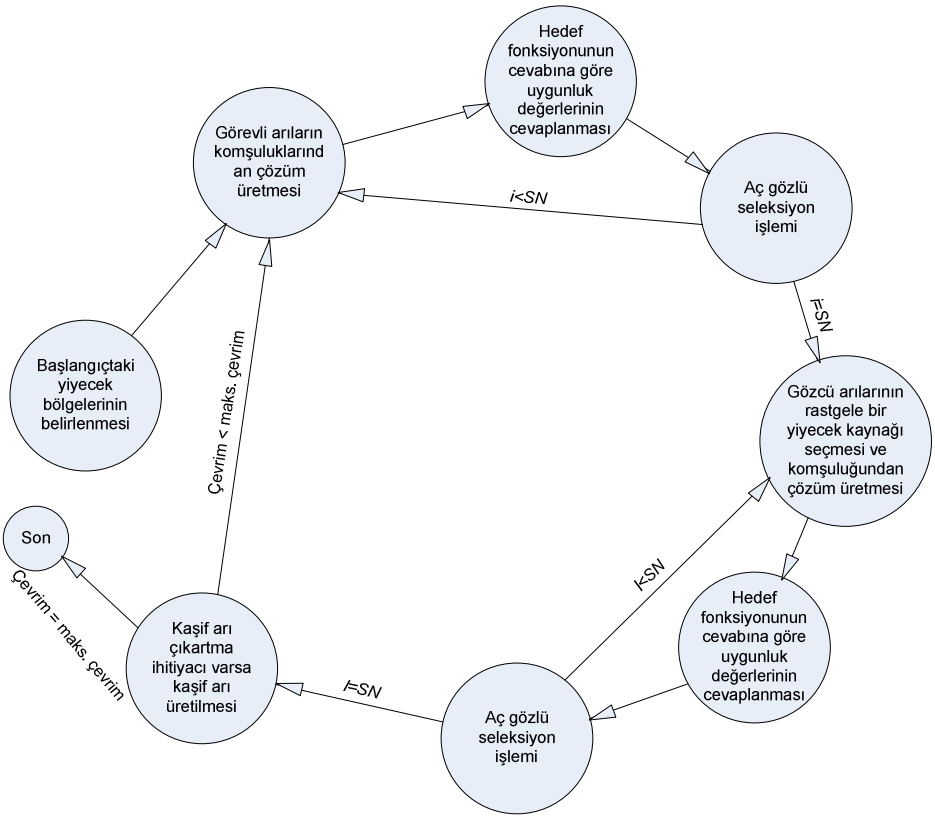
Bu modül görevli ve gözcü arılarının üretmiş olduğu yeni çözümleri uygunluk fonksiyonun cevaplarına göre eski çözümler ile karşılaştırıp en iyi çözümün saklanması görevini yerine getirmektedir. Eğer eski çözümler kullanıcı tarafından girilen bir kontrol parametresinden daha fazla kez yeni çözümlerden daha iyi ise kaşif arı üretilmesini tetikleyecektir.

4.1.7 Kaşif arı üretme modülü

Bir çevrim sonunda tüm görevli ve gözcü arılar arama süreçlerini tamamladıktan sonra çözüm geliştirememe sayaçları *($failure_i$)* kontrol edilir. Bir arının bir kaynaktan faydalanıp faydalanamadığı, yani gidip geldiği kaynağın nektarının tükenip tükenmediği çözüm geliştirememe sayaçları aracılığıyla bilinir. Bir kaynak için çözüm geliştirememe sayacı kullanıcı tarafından belirlenen bir eşik değerinin(*limit*) üzerindeyse, artık bu kaynaktan yeni çözüm üretilemediği ve bu kaynak terk edilip kaşif arılar ile yeni kaynak bulunması gerektiği ortaya çıkmaktadır. Kaşif arı haline gelen işci arı ile rastgele çözüm arama süreci başlar. Oluşacak kaşif görevli arı tarafından tutulan bir yiyecek kaynağı parametrelerinin yeniden üretilmesi işlemidir ve bu işlem başlangıçta oluşturulan rastgele yiyecek kaynakları işlemi ile aynıdır ve Eşitlik 4.1'e göre gerçekleşir. Her bir parametrenin tekrardan oluşturulmasının donanımsal uygulaması ise Şekil 4.2 deki yapı ile aynıdır.

4.1.8 Kontrol modülü

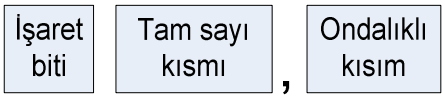
ABC algoritmasına ait tüm modüller ve modüller arasındaki haberleşme kontrol modülü tarafından denetlenir. Kontrol modülü sonlu durum makineleri (Finite State Machine, FSM) metoduyla kodlanmıştır. Kontrol modülüne ait basit bir FSM diyagramı Şekil 4.11 de verilmiştir. Kullanıcı tarafından girilen bütün parametreler gerekli modüllere bu modül tarafından iletilir. Yine kullanıcı tarafından belirlenen maksimum döngü sayısı (Maximum Cycle Number, MCN) ve benzeri döngülerin kontrolü bu modül tarafından gerçekleştirilir.



Şekil 4.11 ABC algoritması için planlana kontrol ünitesine ait sonlu durum makinesi (FSM) diyagramı

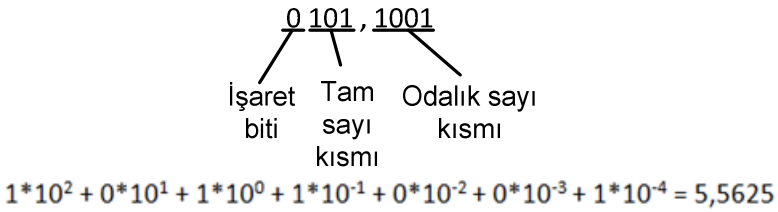
4.2 Donanım Üzerinde Yapılan Aritmetik İşlemler için Kullanılan Sayı Sistemi

ABC algoritması oluşturulurken aritmetik işlemlerin yapılabilmesi için gerçek sayıların ifadesinde sabit noktalı sayı sistemi kullanılmıştır. Bu sayı sistemi ikili sayı sistemi olarak da adlandırılmaktadır. Bu sayı sistemi 3 bölümden oluşup bölümler işaret biti, tam sayı kısmı ve ondalıklı kısımdır (Şekil 4.12).



Şekil 4.12 Sabit noktalı sayı gösterimi

Sabit noktalı sayılarda gerçekte virgül yoktur fakat biz virgül varmış gibi düşünüp öyle işlem yaparız. Örnek vermek gerekirse tam sayı kısmı 3 bit ondalıklı kısmı 4 bit olan 01011001 sabit noktalı sayısının ondalık sayı karşılığı Şekil 4.13 deki gibi hesaplanır.



Şekil 4.13 Sabit noktalı sayıların ondalıklı sayılara çevrilmesi

Negatif sayıları ifade etmek için ikilik tabandaki sayının 2'ye tümleyeni alınır. İki ye tümleyeni alma işleminden sonra işaret biti bir olur.

4.3 Kullanıcı Tarafından Girilen Parametreler

Kullanıcı tarafından girilen bu parametreler sayesinde farklı hedef fonksiyonların denenmesi sırasında ABC algoritması modüllerinin içerisinde değişiklik yapılması ihtiyacını ortadan kaldırmaktadır. Bu sayede her problem için ABC algoritmasını değiştirmek gerekmemektedir. Aynı zamanda ABC algoritmasının çözüm için ne kadar süre çevrim yapılacağı ve ne zaman kaşif arı üretileceği gibi seçimler bu parametreler kullanılarak gerçekleştirilmektedir.

.

- FoodNumber : ABC algoritması için yiyecek kaynağı sayısını belirtir.
- MaxCycles: Algoritmanın ne kadar çevrim yapacağını belirtir
- Limit: Kaynağın tükendiğinin belirlenmesi için kullanılan eşik değeridir. ABC algoritmasının önemli bir kontrol parametrelerinden biridir.
- e: Sistem içinde kullanılan sabit noktalı sayı formatının tam sayı kısmını ifade eder. Bu değer uygulanan problemin çözüm uzayına göre seçilir.

- f: Sistem içinde kullanılan sabit noktalı sayı formatının ondalıklı sayı kısmını ifade eder. Bu değer sistemin çalışması istenen hassasiyete göre seçilir.
- Dim: Optimize edilmek istenen problemin parametre sayısıdır.
- ub: Optimize edilmek istenen problem uzayının üst sınırıdır.
- lb: Optimize edilmek istenen problem uzayının alt sınırıdır.

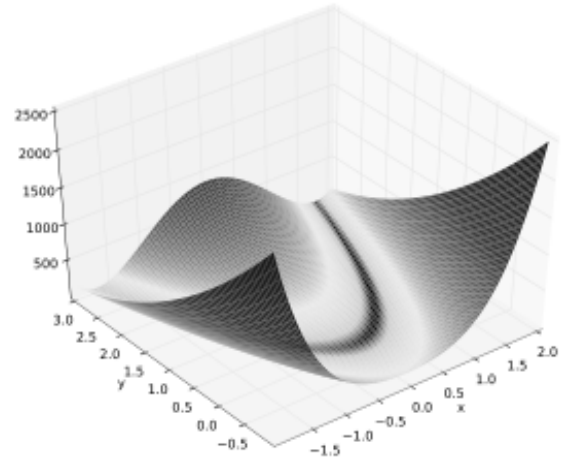
BÖLÜM V

LİTERATÜRDEN SEÇİLMİŞ PROBLEMLER ÜZERİNDE ÖNERİLEN SİSTEM PERFORMANSININ İNCELENMESİ

Oluşturulan ABC algoritması yapısının performansını denemek için literatürde çok kullanılan ve ABC algoritması üzerinde denenmiş zorlu fonksiyonlardan Rosenbrock, OneMax, Sphere ve Rastrigin seçilmiştir. Bu fonksiyonlar literatürden seçilmiş PC tabanlı olarak yapılan çalışmalardan alınan hız, doğruluk gibi sonuçlarla FPGA üzerinde yapılan sonuçlar karşılaştırılmış ve sonuçları çizelgeler ile aşağıdaki bölümlerde gösterilmiştir. FPGA üzerinde oluşturulan modüller VHDL donanım tanımlama dili kullanılarak gerçeklenmiş olup oluşturulan sistem **VIRTEX-5 (XC5VSX50T)** deney seti üzerinde denenmiştir. Sistem çalışma hızı **100 MHz** dir.

5.1. Rosenbrock Fonksiyonu

Rosenbrock fonksiyonu optimizasyon algoritmalarında test amaçlı olarak çok kullanılan fonksiyonlardan biridir. Çalışmada Eşitlik 5.1'de verilmekte olan Rosenbrock fonksiyonunun minimize edilmesi hedeflenmiştir. Bu işlem için Rosenbrock fonksiyonu modül şeklinde yazılıp Şekil 4.1 de gösterildiği gibi hedef fonksiyonu olarak entegre edilmiştir. İki parametreli bir Rosenbrock fonksiyonunu Şekil 5.1 de gösterilmiştir.



Şekil 5.1 İki boyutlu Rosenbrock fonksiyonunun çözüm uzayındaki dağılımı

$$f_{Ros}(x) = \sum_{i=1}^{n-1} \left(100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \right) \quad (5.1)$$

24 parametreli ve çözüm uzayı [-15,15] olan Rosenbrock fonksiyonunun birbirinden bağımsız olarak gerçekleştirilen denemelerin sonuçlarının ortalaması ile literatürde yapılmış olan bazı çalışmalar arasında hız doğruluk bakımından kıyaslama yapılmış ve Çizelge 5.2 de gösterilmiştir. Farklı sayıdaki parametrelere sahip Rosenbrock fonksiyonunun çözümü için oluşturulan ABC algoritmasının FPGA üzerinde kapladığı alan ve hız sonuçları Çizelge 5.1 de gösterildiği gibidir. Bu çizelgede gösterilen uygulamalarda maksimum çevrim sayısı 2000; kaşif arı üretme kriterde 100 olarak seçilmiştir.

Çizelge 5.1 Farklı parametrelerdeki Rosenbrock fonksiyonunun çözümüne ait FPGA üzerinde oluşturulmuş yapının kapladığı alan ve hız sonuçları

Boyut (D)	Aralık	hata	Hız (ms)	register	logic	memory
5	[-15,15]	0	19.3	4118	5967	622
10	[-15,15]	0	19.33	5293	8066	619
15	[-15,15]	0	19.36	6821	11454	618
24	[-15,15]	0.0053	19.38	9270	19183	618

24 parametreli Rosenbrock algoritmasının matlab üzerinde sonuca ulaşma süresi **1150 ms** dir. 24 parametreli Rosenbrock fonksiyonunun donanım üzerinde ABC algoritması ile çözüm süresi PC tabanlı çözümüne göre **60 kat** daha hızlıdır.

Literatürden seçilmiş diğer evrimsel algoritmalarla ABC algoritmasının doğruluk bakımından performansı Çizelge 5.2 de verilmektedir. ABC algoritmasının performansı GA ile [2], PSO, PS-EA ve ABC ile [3]'de tanıtılan çalışmalarında çeşitli test fonksiyonları üzerinde karşılaştırılmıştır. Çizelgede 20 parametreli bir Rosenbrock fonksiyonunun [-15,15] arama uzay aralığında çeşitli evrimsel algoritmalar üzerinde çözümü ve ortalama hataları gösterilmektedir. Çizelgeden de anlaşılacağı üzere ABC algoritması test edilmiş uygulamalar üzerinde aynı şartlar altında diğer yapay zeka optimizasyon algoritmalarından daha iyi sonuç üretmektedir. Aynı zamanda donanım

üzerinde gerçeklenmiş olan ABC algoritması yazılım tabanlı olan ABC algoritması kadar başarılı sonuçlar üretmektedir.

Çizelge 5.2 Rosenbrock fonksiyonu için literatürden seçilmiş evrimsel algoritmaların doğruluk açısından incelenmesi ve ABC algoritması ile karşılaştırılması

	GA[2]	PSO[3]	PS-EA[3]	ABC[3]	Donanım üzerindeki ABC*
Rosenbrock	103.93	77.382	72.452	0.014458	0.0049

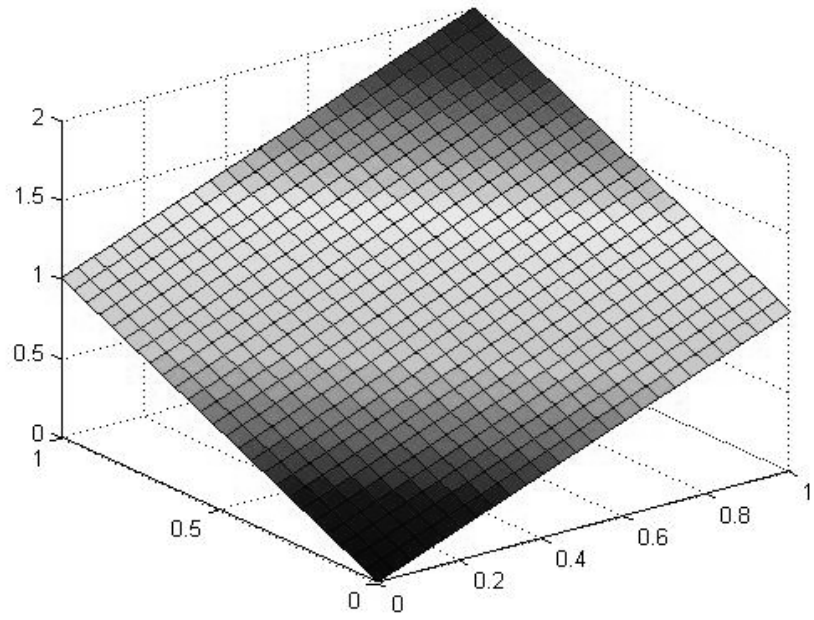
* verilen değer 3 denemenin ortalamasıdır. Karşılaştırma yapılan diğer çalışmalarda 30 denemenin ortalaması verilmiştir.

5.2 OneMax Fonksiyonu

ABC algoritmasının performansının kıyaslanması için kullanılmış olan diğer test fonksiyonu OneMax fonksiyonudur. OneMax fonksiyonu Eşitlik 5.2'de verilmektedir. İki parametreli OneMax fonksiyonunun uzayda dağılımı Şekil 5.2 verilmiştir.

$$f_2\overline{(x)} = \sum_{i=1}^{n} x_i \ , \quad n = 32 \tag{5.2}$$

0<x<1



Şekil 5.2 İki parametreli OneMax fonksiyonunun uzaydaki dağılımı

32 parametreli OneMax fonksiyonunun çözümüne ait FPGA üzerinde oluşturulmuş yapının kapladığı alan ve hız sonuçları Çizelge 5.3'de gösterildiği gibidir. Bu çizelgede gösterilen uygulamada maksimum çevrim sayısı 200; kaşif arı üretme kriteri de 50 olarak seçilmiştir.

Çizelge 5.3 32 parametreli OneMax fonksiyonunun çözümü için FPGA üzerinde oluşturulmuş yapının kapladığı alan ve hız sonuçları

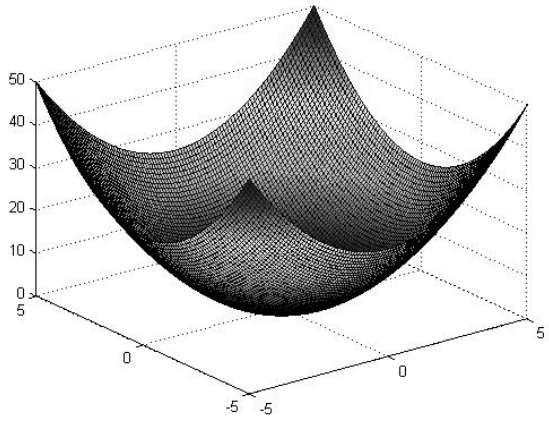
Boyut(D)	Aralık	hata	Hız(ms)	register	logic	memory
32	[0,1]	0.5795	3.72	12144	19632	720

32 parametreli OneMax algoritmasının matlab üzerinde sonuca ulaşma süresi **1410 ms** dir. 32 parametreli OneMax fonksiyonunun donanım üzerinde ABC algoritması ile çözüm süresi PC tabanlı çözümüne göre **380 kat** daha hızlıdır.

5.3 Sphere Fonksiyonu

Denemelerde kullanılmış olan diğer fonksiyon Eşitlik 5.3'de verilmekte olan Sphere fonksiyonudur. İki parametreli Sphere fonksiyonunun uzaydaki dağılımı Şekil 5.3 de gösterilmiştir.

$$f_1(\overline{x}) = \sum_{i=1}^{n} x_i^2 \ , \quad n = 30 \tag{5.3}$$



Şekil 5.3 İki parametreli Sphere fonksiyonunun uzaydaki dağılımı

30 parametreli Sphere fonksiyonunun çözümüne ait FPGA üzerinde oluşturulmuş yapının kapladığı alan ve hız sonuçları Çizelge 5.4'de gösterildiği gibidir. Bu çizelgede gösterilen uygulamada maksimum çevrim sayısı 200 kaşif arı üretme kriterde 50 olarak seçilmiştir.

Çizelge 5.4 30 parametreli Sphere fonksiyonunun çözümü için FPGA üzerinde oluşturulmuş yapının kapladığı alan ve hız sonuçları

Boyut(D)	Aralık	hata	Hız(ms)	register	logic	memory
30	[-1,1]	1.23	3.62	10841	22876	720

30 parametreli Sphere fonksiyonunun matlab üzerinde sonuca ulaşma süresi **1150 ms** dir. 30 parametreli Sphere fonksiyonunun donanım üzerinde ABC algoritması ile çözüm süresi PC tabanlı çözümüne göre **317 kat** daha hızlıdır.

5.4 Rastrig Fonksiyonu

Ölçeklenebilir ve sürekli bir fonksiyon olan Rastrigin fonksiyonu Eşitlik 5.4'de verilmektedir.

$$f_{Ras}(x) = a \cdot n + \sum_{i=1}^{n} \left(x_i^2 - a \cdot Cos(\omega \cdot x_i) \right) \qquad (5.4)$$

$$a = 10, \ \omega = 2\pi$$

$$-5.12 \le x_i \le 5.12$$

Birbirinden bağımsız olarak gerçekleştirilen denemelerin sonuçlarının ortalaması ile ABC algoritmasının performansı GA ile [10], PSO, PS-EA ve ABC ile [11]'de tanıtılan çalışmalarında çeşitli test fonksiyonları üzerinde Çizelge 5.6 da karşılaştırılmıştır. 30 parametreli Rastrigin fonksiyonunun çözümüne ait FPGA üzerinde oluşturulmuş yapının kapladığı alan ve hız sonuçları Çizelge 5.5'de gösterildiği gibidir. Bu çizelgede gösterilen uygulamada maksimum çevrim sayısı 2000; kaşif arı üretme kriteri de 100 olarak seçilmiştir.

Çizelge 5.5 30 parametreli Rastrig fonksiyonunun çözümü için FPGA üzerinde oluşturulmuş yapının kapladığı alan ve hız sonuçları

Boyut(D)	Aralık	hata	Hız (ms)	register	logic	memory
30	[-15,15]	0	104	13143	26674	1025

30 parametreli Rastrigin fonksiyonunun matlab üzerinde sonuca ulaşma süresi **13330 ms** dir. 30 parametreli Rastrigin fonksiyonunun donanım üzerinde ABC algoritması ile çözüm süresi PC tabanlı çözümüne göre **128 kat** daha hızlıdır.

Çizelge 5.6 de 30 parametreli bir Rastrigin fonksiyonunun [-15,15] arama uzay aralığında çeşitli evrimsel algoritmalar üzerinde çözümü ve ortalama hataları gösterilmektedir. Çizelgeden de anlaşılacağı üzere ABC algoritması diğer evrimsel algoritmalardan daha iyi sonuç üretmektedir. Aynı zamanda donanım üzerinde gerçeklenmiş olan ABC algoritması yazılım tabanlı olan ABC algoritması kadar başarılı sonuçlar üretmektedir.

Çizelge 5.6 Rastrig fonksiyonu için literatürden seçilmiş evrimsel algoritmaların doğruluk açısından incelenmesi ve ABC algoritması ile karşılaştırılması

	GA[2]	PSO[3]	PS-EA[3]	ABC[3]	Donanım üzerindeki ABC*
Rastrig	10.4388	32.476	3.0527	0	0

* verilen değer 3 denemenin ortalamasıdır. Karşılaştırma yapılan diğer çalışmalarda 30 denemenin ortalaması verilmiştir.

5.5 Bölüm Çıktıları

Oluşturulan ABC algoritma yapısının performansını denemek için literatürde çok kullanılan ve ABC algoritması üzerinde denenmiş zor fonksiyonlar kullanılmış ve literatürde yapılmış olan çalışmalar üzeriden hız ve ortalama hata bakımından değerlendirilmiştir. Alınan sonuçlar göstermiştir ki diğer evrimsel algoritmalarından daha güçlü olan ABC algoritmasının donanım üzerinde uygulanması zorlu problemlerin çözümünde daha hızlı aynı zamanda PC tabanlı uygulamalardaki doğrulukta çözümler sunmaktadır. ABC algoritmasının birkaç ms de çözüm üretebilmesi bakımından gerçek zamanlı uygulamalarda donanım üzerinde uygulanması önemlidir.

BÖLÜM VI

DONANIM ÜZERİNDE YAPAY SİNİR AĞLARININ ABC ALGORİTMASI İLE EĞİTİLMESİ

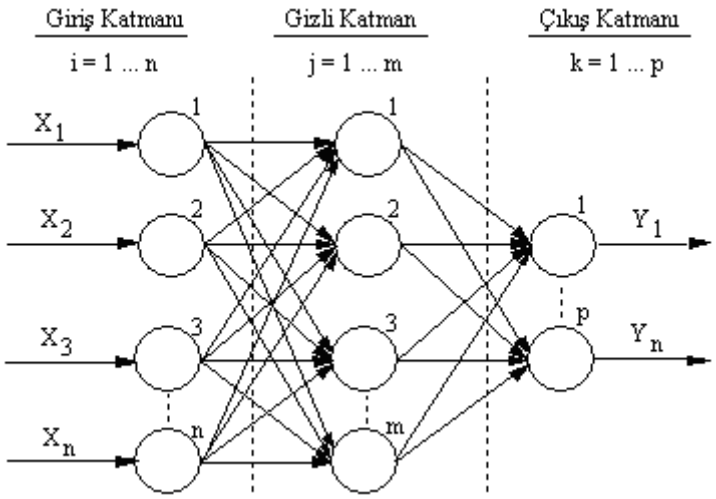
Yapay Sinir Ağları (YSA) biyolojik beyinin öğrenme ve bilgi saklama yeteneklerinin benzetilmeye çalışıldığı yapay zeka tekniğidir. YSA'da bilgileri depolayan bağlantı ağırlıkları nöronları paralel ve ardışık bir biçimde birleştirirler. Oldukça zor bir optimizasyon problemi olan Yapay sinir ağlarının öğrenmesi süreci ağ yapısında ki ağırlıklara uygun değerlerin bulunması işlemidir. Bir çok çalışmanın yapıldığı bu alanda ilk çalışmalar optimizasyon tekniklerine dayalı iken genel en iyileştirici algoritmaların geliştirilmesi ile bu metotlar da yapay sinir ağlarının eğitiminde yaygın olarak kullanılmaya başlanmıştır. Bu yüksek lisans tez çalışmasında, yeni bir genel araştırma tekniği olan sürü zekasına dayalı Yapay Arı Kolonisi (ABC) algoritmasının donanınım üzerindeki performansını ölçmek için YSA eğitimi önerilmiş ve eğitim süresi ve performansı birkaç problem için ölçülmüştür. Tez çalışmasında ABC algoritması ilk olarak basit bir exor işlemini eğitim için kullanılmış ve ardından literatürden seçilmiş olan bir plaka bulma algoritmasında kullanılan YSA'nın yeniden eğitimi ve bilgi setleri üzerinden performansı ölçülmüştür.

6.1 Yapay Sinir Ağları

Yapay sinir ağları (YSA), insan beyninin bilgi işleme teknolojisinden esinlenerek geliştirilmiş bir bilgiişlem teknolojisidir. YSA ile basit biyolojik sinir sisteminin çalışma şekli simüle edilir (benzetilir). Simüle edilen sinir hücreleri nöronlar içerirler ve bu nöronlar çeşitli şekillerde birbirlerine bağlanarak ağı oluştururlar. Bu ağlar öğrenme, hafızaya alma ve veriler arasındaki ilişkiyi ortaya çıkarma kapasitesine sahiptirler. Diğer bir ifadeyle, YSA'lar, normalde bir insanın düşünme ve gözlemlemeye yönelik doğal yeteneklerini gerektiren problemlere çözüm üretmektedir. Bir insanın, düşünme ve gözlemleme yeteneklerini gerektiren problemlere yönelik çözümler üretebilmesinin temel sebebi ise insan beyninin ve dolayısıyla insanın sahip olduğu yaşayarak veya deneyerek öğrenme yeteneğidir [30].

Biyolojik sistemlerde öğrenme, nöronlar arasındaki sinaptik (*synaptic*) bağlantıların ayarlanması ile olur. Yani, insanlar doğumlarından itibaren bir *yaşayarak öğrenme* süreci içerisine girerler. Bu süreç içinde beyin sürekli bir gelişme göstermektedir. Yaşayıp tecrübe ettikçe sinaptik bağlantılar ayarlanır ve hatta yeni bağlantılar oluşur. Bu sayede öğrenme gerçekleşir. Bu durum YSA için de geçerlidir. Öğrenme, eğitme yoluyla örnekler kullanarak olur; başka bir deyişle, gerçekleşme girdi/çıktı verilerinin işlenmesiyle, yani eğitme algoritmasının bu verileri kullanarak bağlantı ağırlıklarını (*weights of the synapses*) bir yakınsama sağlanana kadar, tekrar tekrar ayarlamasıyla olur [30].

YSA'lar, ağırlıklandırılmış şekilde birbirlerine bağlanmış birçok işlem biriminden (nöronlar) oluşan matematiksel sistemlerdir. Bir işlem birimi, aslında sık sık transfer fonksiyonu olarak anılan bir Eşitliktir. Bu işlem birimi, diğer nöronlardan sinyalleri alır; bunları birleştirir, dönüştürür ve sayısal bir sonuç ortaya çıkartır. Genelde, işlem birimleri kabaca gerçek nöronlara karşılık gelirler ve bir ağ içinde birbirlerine bağlanırlar; bu yapı da sinir ağlarını oluşturmaktadır(Şekil 6.1).



Şekil 6.1 Yapay sinir ağı modeli ve katmanları

Sinirsel (*neural*) hesaplamanın merkezinde dağıtılmış, adaptif ve doğrusal olmayan işlem kavramları vardır. YSA'lar, geleneksel işlemcilerden farklı şekilde işlem yapmaktadırlar. Geleneksel işlemcilerde, tek bir merkezi işlem birimi her hareketi sırasıyla gerçekleştirir. YSA'lar ise herbiri büyük bir problemin bir parçası ile ilgilenen,

çok sayıda basit işlem birimlerinden oluşmaktadır. En basit şekilde, bir işlem birimi, bir girdiyi bir ağırlık kümesi ile ağırlıklandırır, doğrusal olmayan bir şekilde dönüşümünü sağlar ve bir çıktı değeri oluşturur. İlk bakışta, işlem birimlerinin çalışma şekli yanıltıcı şekilde basittir. Sinirsel hesaplamanın gücü, toplam işlem yükünü paylaşan işlem birimlerinin birbirleri arasındaki yoğun bağlantı yapısından gelmektedir. Bu sistemlerde geri yayılım metoduyla daha sağlıklı öğrenme sağlanmaktadır.

Çoğu YSA'da, benzer karakteristiğe sahip nöronlar tabakalar halinde yapılandırılırlar ve transfer fonksiyonları eş zamanlı olarak çalıştırılırlar. Hemen hemen tüm ağlar, veri alan nöronlara ve çıktı üreten nöronlara sahiptirler.

6.1.1 Yapay sinir ağlarının yapısı

Yapay sinir ağları yapay sinir hücrelerinin birbirine bağlanmasıyla oluşan yapılardır. Yapay sinir ağları Şekil 6.1 de de görüldüğü üzere üç ana bölümde incelenir; giriş, ara ve çıkış katmanları.

Giriş Katmanı

Yapay sinir ağına dış dünyadan girdilerin geldiği katmandır. Bu katmanda dış dünyadan gelecek giriş sayısı kadar nöron bulunmasına rağmen genelde girdiler herhangi bir işleme uğramadan alt katmanlara iletilmektedir.

Ara Katmanı(Gizli Katman)

Giriş katmanından çıkan bilgiler bu katmana gelir. Ara katman sayısı ağdan ağa değişebilir. Bazı yapay sinir ağlarında ara katman bulunmadığı gibi bazı yapay sinir ağlarında ise birden fazla ara katman bulunmaktadır. Ara katmanlardaki nöron sayıları giriş ve çıkış sayısından bağımsızdır. Birden fazla ara katman olan ağlarda ara katmanların kendi aralarındaki nöron sayıları da farklı olabilir. Ara katmanların ve bu katmanlardaki nöronların sayısının artması hesaplama karmaşıklığını ve süresini arttırmasına rağmen yapay sinir ağının daha karmaşık problemlerin çözümünde de kullanılabilmesini sağlar.

Çıkış Katmanı

Ara katmanlardan gelen bilgileri işleyerek ağın girdi katmanından gelen verilere karşılık olan çıktıları üreten katmandır. Bu katmanda üretilen çıktılar dış dünyaya gönderilir. Geri beslemeli ağlarda bu katmanda üretilen çıktı kullanılarak ağın yeni ağırlık değerleri hesaplanır.

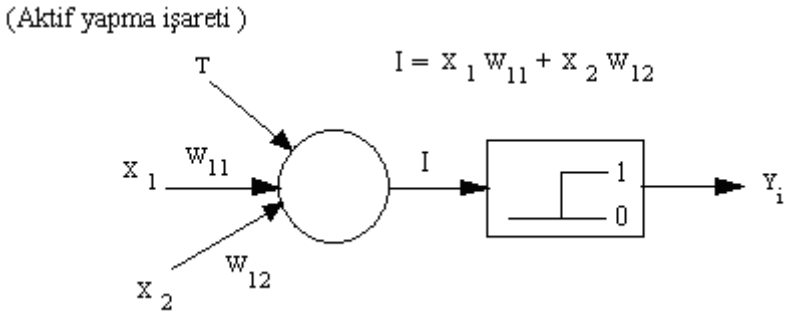
6.1.2 Nöron modeli

Örnek bir nöron modeli şu şekilde açıklanmaktadır [30]:

Elmas, Ç., Yapay Zeka Uygulamaları, Seçkin Yayıncılık, İstanbul 2007

"Nöronlar çok sayıda dentritleri vasıtasıyla giriş uyarısını alır. Dentritlerce alınan bir giriş, harekete geçirici (tetikleyici) veya yasaklayıcı olabilir. Girişler toplanır ve nöron gövdesine yerleştirilir. Bu giriş, belirli bir eşik değerini aştığı zaman, hücre diğer hücrelere aksonu vasıtasıyla bir etki iletir. Nöron davranışı bütün yönleriyle anlaşılmaktan uzak olmasına rağmen, bu basit anlatım yapay nöron için bir model oluşturur.

Bir nöron/işlem elemanı (Şekil 6.2), kendine gelen girişleri toplayan ve bir aktivasyon fonksiyonu işlemiyle eşik değerinin ölçümü sonrasında bir çıkış üreten bir işlem elemanıdır. Bir eşik birimi olarak nöron snapslarındaki işaretleri alır ve hepsini toplar. Eğer toplanan işaret gücü, eşiği geçecek kadar güçlü ise, diğer nöronları ve dentritleri uyarmak üzere akson boyunca bir işaret gönderilir. Kesişen dentritlerde oluşan snapslarca tutulan bütün işaretleri soma toplar. Toplam işaret daha sonra nöronun iç eşik değeri ile karşılaştırılır ve eşik değeri aşmışsa aksona bir işaret yayar. YSA bu basit nöronların (düğümlerin ya da ünitelerin) bağlanarak bir ağa dönüştürülmesiyle meydana getirilir."



Şekil 6.2 Bir nöronun iç yapısı

6.1.3 Ağ tipleri

Üç çeşit ağ tipi vardır:

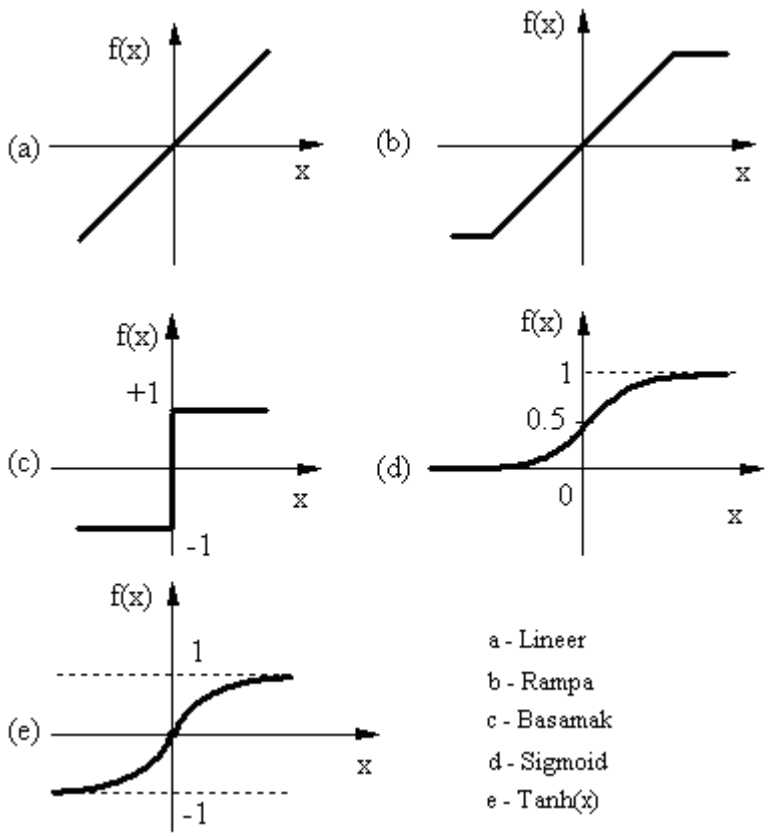
Ileri Beslemeli Ağ: Herbir katmandaki hücreler sadece önceki katmanın hücrelerince beslenir ve dolayısıyle bilgi akışı sadece ileriye doğrudur.

Kaskat Bağlı Ağ: Hücreler sadece önceki katmanlardaki hücrelerce beslenir.

Geri Beslemeli Ağ: En az bir hücre sonraki katmanlardaki hücrelerce beslenir. Bir katmandan diğerine veya kendisine doğru uzanan ağırlıklar yoluyla geri besleme bağlantılarına izin veren daha genel bir ağ yapısı gösterir.

6.1.4 Eşik fonksiyonları

Transfer veya işaret fonksiyonları olarak da adlandırılan eşik (aktivasyon) fonksiyonları, muhtemel sonsuz giriş kümesine sahip işlem elemanlarından önceden belirlenmiş sınırlar içinde çıkışlar üretirler. Beş tane yaygın olarak kullanılan eşik fonksiyonu vardır. Bunlar lineer, rampa, basamak, sigmoid ve tanh(x) fonksiyonlarıdır. Şekil 6.3'de bu fonksiyonlar gösterilmiştir. Şekil 6.3a'de gösterilen lineer fonksiyonun matematiksel değeri Eşitlik 6.1 deki gibidir.



Şekil 6.3 Sıkça kullanılan eşik fonksiyonları

$$f(x) = a \cdot x \tag{6.1}$$

a işlem elemanının *x* aktivitesini ayarlayan reel değerli bir sabittir. Lineer fonksiyon (-t, +t) sınırları arasında kısıtlandığında Şekil 6.3b'deki rampa eşik fonksiyonu olur (Eşitlik 6.2).

$$f(x) = \begin{cases} +t & : x \geq t \\ x & : -t < x < t \\ -t & : x \leq -t \end{cases} \tag{6.2}$$

+t/-t işlem elemanının maksimumu/minimumu çoğu zaman doyma seviyesi olarak adlandırılan çıkış değeridir. Şekil 6.3c, basamak eşik fonksiyonunu gösterir ve matematiksel değeri Eşitlik 6.3 de verildiği gibidir.

$$f(x) = \begin{cases} -t, & x < 0 \\ t, & x \geq 0 \end{cases} \tag{6.3}$$

Diğer bir eşik fonksiyonu ise sigmoid fonksiyonudur ve bu tez çalışmasında eşik fonksiyonu olarak kullanılmıştır. Şekil 6.3d'de gösterilen yatık S biçimindeki sigmoid fonksiyonu; seviyeli, doğrusal olmayan(non-lineer) çıkış veren, sınırlı bir fonksiyondur ve matematiksel değeri Eşitlik 6.4 de verilmiştir.

$$f(x)= \frac{1}{1+e^{-x}} \tag{6.4}$$

Son olarak Şekil 6.3e'de ise tanh(x) fonksiyonu görülmektedir ve matematiksel değeri Eşitlik 6.5 de görüldüğü gibidir.

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{6.5}$$

Her işlem elemanı kendisine gelen bir yerel veriye göre, kendisini ayarlayarak bütün YSA nın bilgi bölgesini öğrenmesini sağlar. Yukarıdaki transfer fonksiyonlarını kullanabilmek için, giriş verilerinin gerçek değerlerinin "0" ile "1" arasındaki bir reel sayıya dönüştürülmesi (normalizasyon) gerekir.

6.2 YSA Eğitme Algoritmaları

Eğitme algoritmaları YSA lar için çok elzemdir. Eğitme algoritması, çözülmek istenen problemin özelliğine göre öğrenme kuralının YSA ya nasıl adapte edileceğini belirtir. Kullanımı yaygın olan üç çeşit eğitme algoritması vardır.

- *Öğreticili eğitme (Supervised training):* Öğreticili eğitmede, eğitme örneklerinin tamamı için (x_1, x_2, …, x_n) şeklindeki giriş vektörünün; (y_1, y_2, …, y_n) şeklindeki çıkış vektörü, tam ve doğru olarak bilinmektedir. Herbir (x_1, y_1), (x_2, y_2), …., (x_n, y_n) çifti için, ağ doğru sonuçları verecek şekilde, seçilen bir öğrenme kuralı yardımıyla eğitilir.

- *Skor ile eğitme(Graded training):* Skor ile eğitmede giriş işaretlerine karşılık gelen çıkış işaretleri tam olarak bilinmemektedir. Çıkış işareti yerine skor verilir ve ağın değerlendirmesi yapılır. Özellikle kontrol uygulamaları için idealdir. Çeşitli maliyet (cost) fonksiyonları kullanılır.

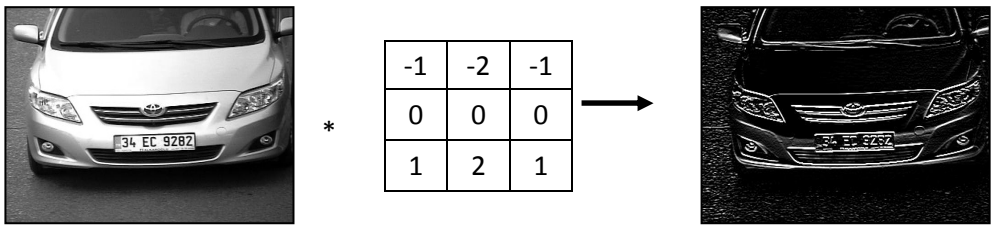
- *Kendini düzenleme ile eğitme(Self-organization training):* Probleme ait veri girişlerine karşın çıkışlar mevcut değildir. Bu yüzden bu tür bir eğitme, giriş verilerini gruplandırarak eğittikten sonra verilen herhangi bir girişin eğitme sınıflarından hangi sınıfa ait olduğunu gösterebilir. Kendini düzenleyen ağ, giriş işaretine göre kendini düzenleyerek organize eder. Bu eğitme tipi olasılık yoğunluk fonksiyonlarına, sınıflandırma ve şekil tanıma problemlerine uygulanabilir.

6.3 YSA'nın Eğitimi için Kullanılan Data Setinin Hazırlanması

ABC algoritmasının performans analizi için kullanılacak olan YSA eğitim problemi için AVCI G. ve ark. [16] tarafından yapılmış plaka tanıma çalışmasında kullanılan YSA'nın tekrar eğitilmesi ele alınmıştır. Buradaki YSA'nın eğitim işleminde kullanılan resimler Şekil 6.4 de verilmiştir.

Şekil 6.4 YSA eğitiminde kullanılmış özniteliklerin elde edildiği örnek imgeler

Her bir araba resmi Şekil 6.5 de görülün yatay sobel filtresinden geçirilerek yatay kenarları elde edilmiştir. Elde edilen bu imgeden çıkartılan öznitelikler aşağıdaki bölümlerde anlatılmaktadır.



Şekil 6.5 Yatay sobel filtresi ile yatay kenarların bulunması

6.3.1 Özniteliklerin çıkarılması

Plaka tanıma sisteminde, plakanın yerinin tespiti için plaka olan ve olmayan bölgelerin ayırt edilmesi gerekmektedir. Plaka aday bölgesinin plaka olup olmadığının tespiti için aday bölgeye ait öznitelikler çıkartılmalıdır. Bu çalışmada plaka bölgesinin tespiti için 8 öznitelik çıkarılmıştır. Elde edilen bu özniteliklerden hangilerinin bilgi yoğun olduğunu ve sınıflandırıcı başarımını artırıcı bir etkiye sahip olduğunu tespit etmek için bir öznitelik seçim algoritması kullanılmıştır. Elde edilen sonuçlardan hareketle dört öznitelik kullanımının yüksek başarım için yeterli olduğu görülmüştür. Kullanılacak olan öznitelikler aşağıda verilmiştir.

a) Ortalama Değer :

İmge içinden alınan plaka aday bölgesinde, siyah piksel sayısının toplam piksel değerine oranı ile bulunur (Eşitlik 6.6).

$$Ortalama\ değer = \frac{Toplam\ siyah\ piksel\ sayısı}{Toplam\ piksel\ sayısı} \tag{6.6}$$

Bu öznitelikle plaka bölgesinde kenarların daha belirgin olması nedeniyle beyaz piksel değerinin plaka olmayan bölgelere göre daha kolayca ayırt edilebileceği düşünülmüştür.

b) Varyans :

Aday bölge içinde, ortalama bir dağılımın merkez noktasını bulmak için kullanılır (Eşitlik 6.7).

$$varyans = \frac{1}{N}\sum_{i=1}^{N}(imge(i) - ortalama\ değer)^2 \quad (6.7)$$

c) Diğer İstatiksel Öznitelikler

Plaka tespitinde kullanılan, diğer öznitelikler Şekil 6.6 da görüldüğü gibi aday bölgeyi üç parçada incelemektedir. Bu parçalar aday bölgenin %25, %50 ve %25'i şeklinde bölünmektedir. Bu aday bölgelerin her birine ait siyah piksel değerleri hesaplanır. Bu özniteliklerin ilki B bölgesine ait siyah piksel değerinden, A ve C bölgelerine ait siyah piksel değerlerinin çıkarılıp toplam piksel değerine bölünmesiyle elde edilir (Eşitlik 6.8). Diğer öznitelik ise C bölgesine ait siyah piksel değerinin A bölgesine ait siyah piksel değerine bölünmesiyle elde edilir (Eşitlik 6.9).

A = % 25
B = % 50
C = % 25

Şekil 6.6 Plaka aday bölgesinin istatiksel parçalara bölünmesi

$$Rc = \frac{siyah\_toplam\_B - siyah\_toplam\_A - siyah\_toplam\_c}{toplam\_pixel} \quad (6.8)$$

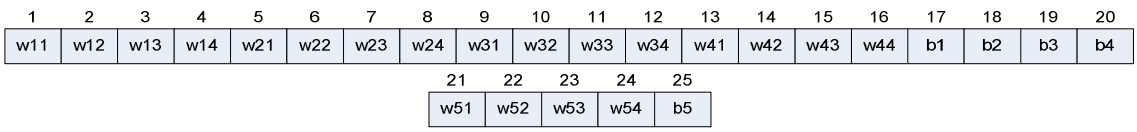
$$Rl = \frac{siyah\_toplam\_C}{siyah\_toplam\_A} \quad (6.9)$$

Aday bölge içinde, ilk olarak ortalama değerin hesaplanması için siyah piksel değerleri tespit edilir. Aynı zamanda aday bölge satır sayısına göre, A, B ve C bölgelerindeki siyah piksel değerleri hesaplanır. Aday bölgenin tamamındaki siyah piksellerin değerinin aday bölgeye ait tüm piksellere bölümü ile ortalama değer hesaplanır. *Rc* ve *Rl* öznitelikleri ise aday bölgenin satir değerleriyle bulunduktan sonra B bölgesine ait siyah piksellerden A ve C bölgesine ait siyah piksel değerlerinin çıkartılıp tüm piksel

değerlerine bölünmesiyle *Rc*, C bölgesine ait piksel değerlerinin A bölgesine ait piksel değerlerine bölünmesiyle de *Rl* özniteliği elde edilir. Daha sonra ortalama değer kullanılarak varyans hesaplanır ve aday bölgenin plaka olup olmadığını tespit amaçla YSA'ya giriş olarak verilmeye hazır hale getirilir.

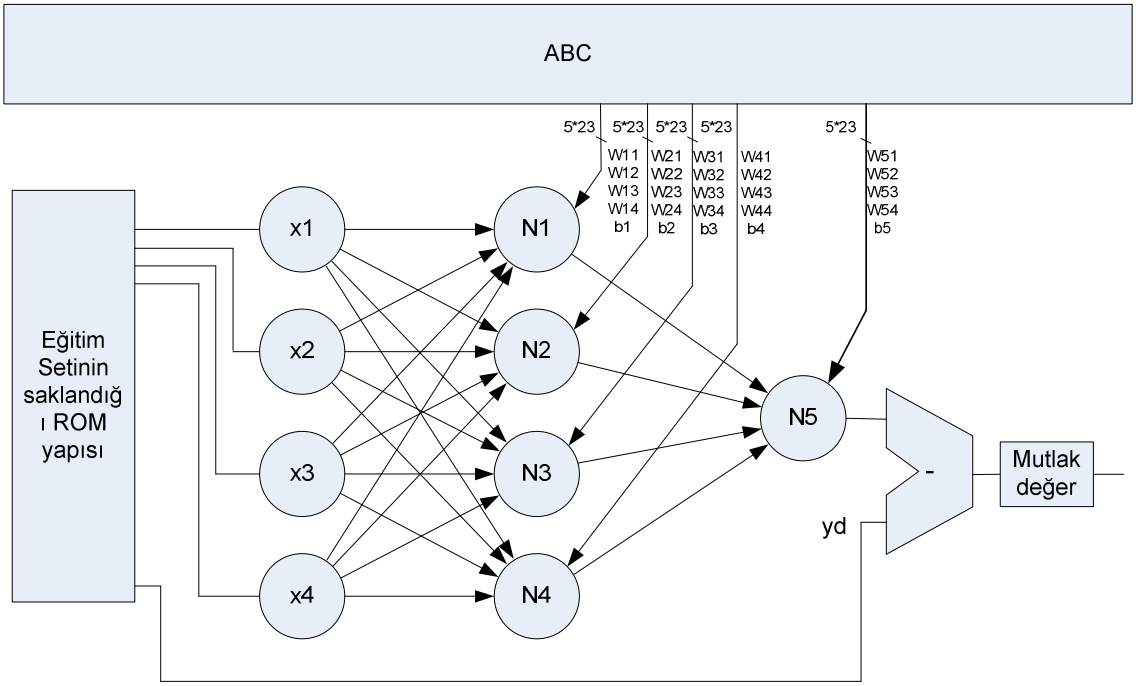
6.4 Donanım Üzerinde YSA'nın Oluşturulması

ABC algoritması kullanarak YSA eğitiminde de amaç ağırlıkların en uygun şekilde hesaplanması ve eğitim ve test aşamasında minimum hata ile çözüm elde edilmesidir. Arama uzayını yiyecek kaynakları olarak düşündüğümüzde bu modelde YSA'nın ağırlıkları ABC algoritması için yiyecek kaynağı anlamına gelmektedir. Oluşturulacak YSA yapısında 5 adet nöron (4 adet saklı katmanda, 1 adet çıkış katmanında) olup her nöronun da 5 adet ağırlığı bulunmaktadır. Buda optimize edilecek toplam 25 adet ağırlık demektir. YSA eğitimini ABC algoritması için hedef fonksiyon olarak düşünürsek 25 parametreli bir vektör ortaya çıkar. YSA ağırlıklarının ABC'nin yiyecek kaynağı koordinatları şeklinde ifade edilişi Şekil 6.7 de gösterilmiştir. Aynı zamanda bu vektör hedef fonksiyonunun (YSA modülünün) giriş vektörüdür.



Şekil 6.7 YSA'nın ağırlıklarının ABC algoritmasının çözüm vektörü şeklinde gösterimi

Şekil 6.7 de gösterilen 1 den 25 e kadar olan sayılar ABC algoritmasının çözüm uzayı parametreleri olup altındaki kutulardaki değerler ise bu parametrelere karşılık olarak eğitim sonucu bulunacak olan YSA ağırlıklarını göstermektedir. Bu ağırlıkların YSA modeli üzerindeki yerleri Şekil 6.8 de gösterildiği gibidir.



Şekil 6.8 FPGA içinde oluşturulan YSA yapısı

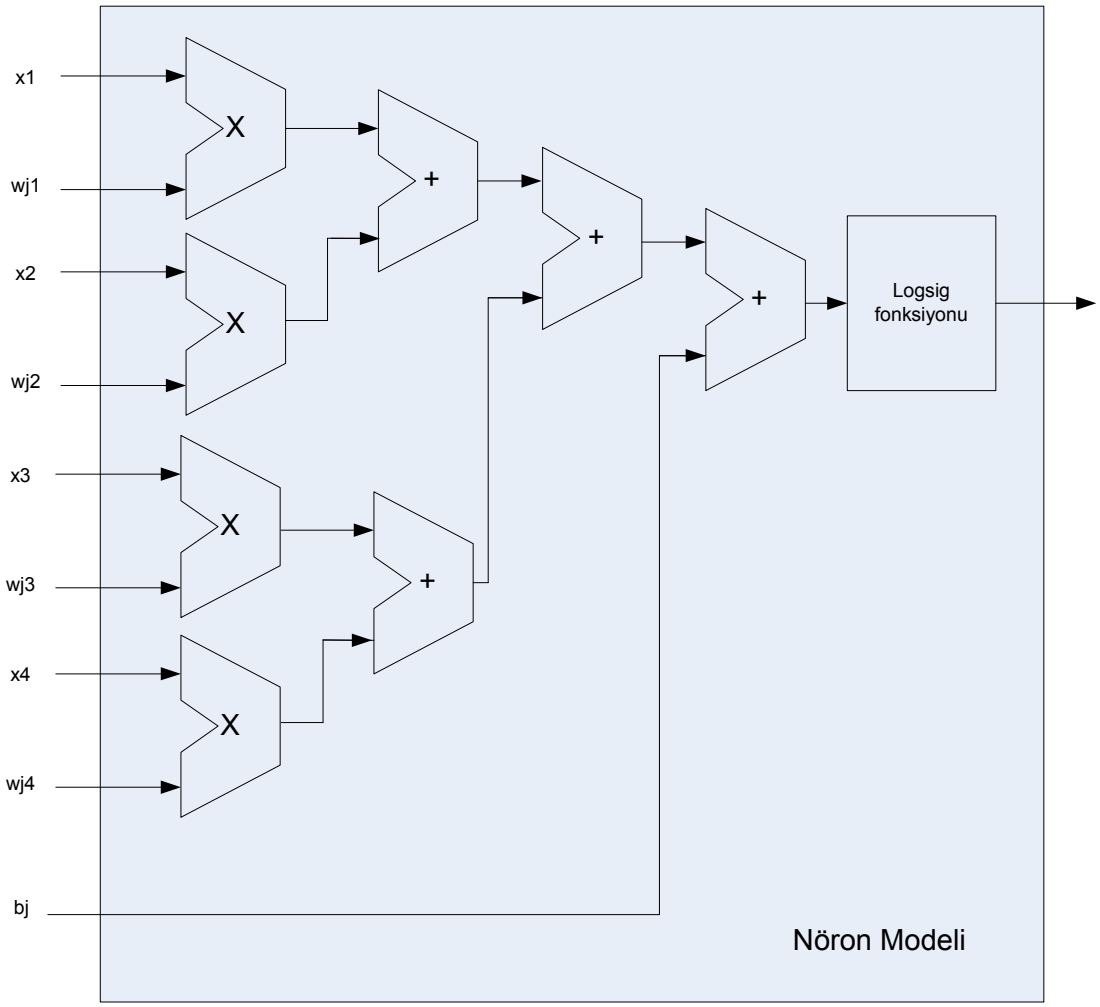
Şekil 6.8 de gösterilern ara katmandaki nöronlar (N1, N2, N3, N4) giriş katmandaki öznitelik nöronları(x1,x2,x3,x4)'dan aldığı bilgileri ABC algoritmasının YSA modülüne giriş vektörü olarak verdiği ağırlıklar (w11,..,w54,b1,..,b5) ile çarpıp bunları Eşitlik 6.10'a göre hesaplamaktadır.

$$f(nöron_j) = logsig((\sum_{i=1}^{4} x_i * w_{ji}) + b_j) \quad (6.10)$$

Eşitlik 6.10 da gösterilen *f(nöron$_j$) j.* nöron'nun çıkış değeridir. x_i *i.* özniteliğin değeri w_{ji} ve b_j ise *j.* nöronun *i.* ağırlıklarıdır. Oluşturulan yapı Şekil 6.9 da gösterilmektedir.

Şekil 6.8 de gösterilen YSA yapısındaki *x1,x2,x3,x4* öznitelikleri ifade edip bu öznitelikler YSA modülünün içindeki eğitim setinin saklandığı ROM yapısından okunmaktadır. Bu ROM yapısı oluşturulurken Şekil 6.4 de gösterilen örnek resimler gibi bir resim seti üzerinden 959 adet plaka bölgesi ve 1077 adet plaka bölgesi olmayan bölge seçilip toplam 2036 adet imgenin her birinden 4 adet öznitelik çıkarılmıştır. Bu oluşturulan öznitelikler setinden rastgele seçilmek üzere 200 adet plaka bölgesi ve 200 adet plaka bölgesi olmayan toplam 400 adet öznitelikler seti oluşturulup YSA'yı

eğitmek için bilgisayar ortamında Xilinx ISE programının arayüzü ile FPGA'nın içinde oluşturulan bir dahili ROM yapısına saklanmıştır.

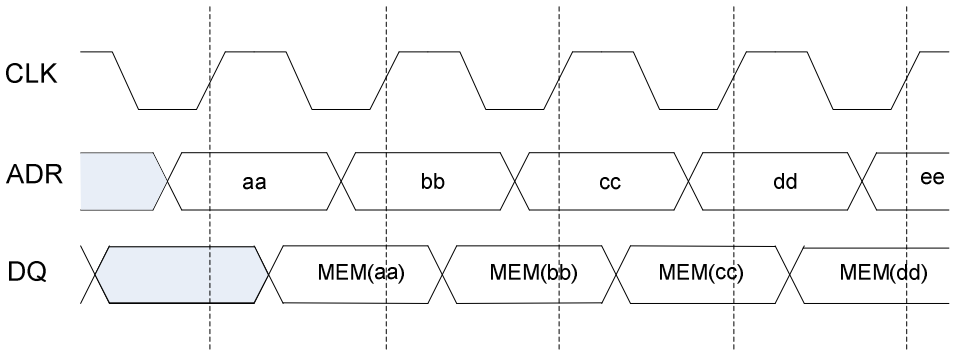


Şekil 6.9 FPGA içerisinde gerçeklenmiş nöronun yapısı

Yine aynı zamanda aynı bilgi setinden 50 adet plaka bölgesi ve 50 adet de plaka olmayan bölgeden seçilmiş 100 adet öznitelikler seti eğitim işlemi bitiminde elde edilmiş ağırlıklarla YSA'yı test etmek için yine FPGA'nın içinde oluşturulan bir dahili ROM yapısına saklanmıştır.

Eğitimde kullanılacak YSA 4 girişi, 4 saklı katmanı ve 1 çıkışı olan bir yapıya sahiptir. *x1,x2,x3,x4* ile gösterilen nöronlar giriş nöronlarını aynı zamanda öznitelikleri ifade etmektedir. Giriş nöronları Şekil 6.8 de de görüldüğü üzere eğitim setinin saklandığı ROM yapısından öznitelikleri okuyup ara katmandaki nöronlara aktarmaktadır.

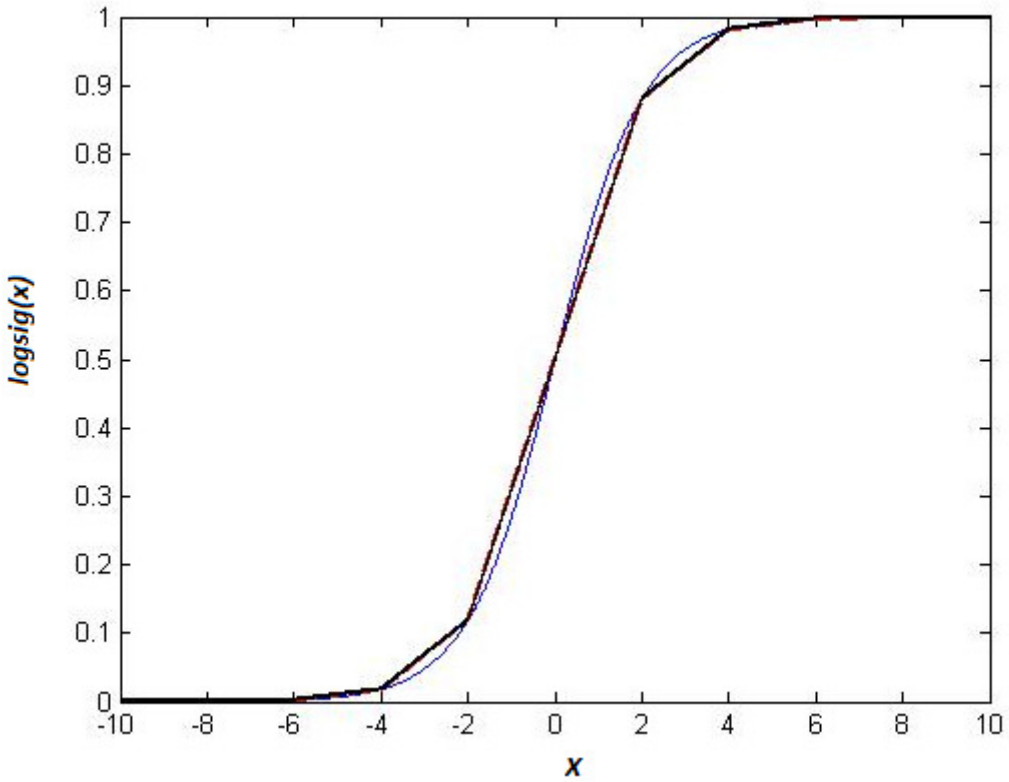
Bu dahili ROM yapıları FPGA içerisinde sadece bu iş için ayrılmış alanları kullanıp ABC algoritması için gerekli donanım kaynaklarını harcamamaktadır. Dahili ROM yapılarına bilgi setleri bilgisayar üzerinde Xilinx ISE programının bir arayüzü ile bir kereliğine mahsus gömülmüştür. Şekil 6.10 de de görüldüğü üzere bir ROM bölgesinden bir datanın okunması bir saat darbesi kadar sürmektedir.



Şekil 6.10 ROM'dan data okuma zaman çizelgesi

Oluşturulan nöron yapısında kullanılan aktivasyon fonksiyonu logsig fonksiyonudur ve eşitlik 6.4 da gösterildiği gibidir.

ABC algoritması en iyi ağırlıkları bulmak için her bir çevrimde yiyecek kaynağının 2 katı kadar YSA ağını eldeki ağırlıklarla deneyecektir. Bir YSA da 5 tane nöron (4 adet saklı katmanda, 1 adet çıkış katmanında) olduğu için logsig fonksiyonunu da yiyecek kaynağının 10 katı kadar çalıştıracaktır. Logsig fonksiyonunda bulunan bölme işlemi kullanılan sayı sistemine bağlı olmakla birlikte önemli derecede zaman harcamaktadır. Bu yüzden bölme işleminden kurtulmak için parçalı doğrusal yaklaşım yöntemi kullanılmıştır. Şekil 6.11 da örnek bir [-10,10] aralığındaki logsig fonksiyonunun 10 eşit parçaya bölünmüş hali görülmektedir.

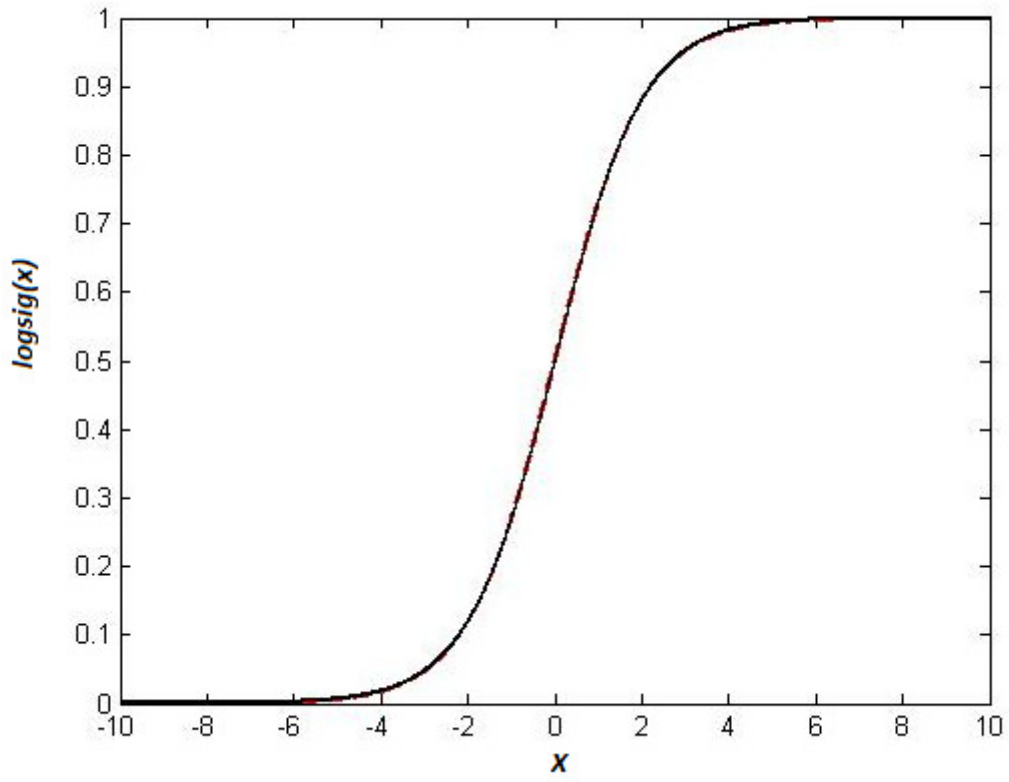


Şekil 6.11 Logsig fonksiyonunun parçalı doğrusal yaklaşım yöntemiyle 10 eşit parçaya bölünmüş hali

Şekil 6.11 da gösterilen 10 segmentden oluşan parçalı doğrusal yaklaşım ile gerçek logsig fonksiyonu karşılaştırıldığında ortalama hatası 2.2130e-004 iken maksimum hatasıda 0.0409 olarak hesaplanmıştır.

Yaklaşımdaki hata miktarını daha da azaltmak için parçalı doğrusal yaklaşım yönteminde gercek logsig fonksiyonu 80 eşit parçaya bölünüp her bir parçaya ait Eşitlik 6.11de gösterilen m ve n değerleri ROM(m) ve ROM(n) e saklanmıştır. Böylelikle giriş bilgisine göre ROM'dan uygun değerler çekilip Eşitlik 6.11'e göre çıkış hesaplanmaktadır. Parçalı doğrusal yaklaşımlı logsig fonksiyonu ile gerçek logsig fonksiyonu Şekil 6.12 de görülmektedir ve ikisi arasında büyük bir fark olmadığı anlaşılmaktadır. Şekil 6.12 de gösterilen parçalı doğrusal yaklaşım ile 80 eşit parçaya bölünmüş bir logsig fonksiyonu olup gerçek logsig fonksiyonu ile karşılaştırıldığında ortalama hatası 5.4123e-008; maksimum hatasıda 7.4848e-004 çıkmaktadır.

$$f(i) = m * x(i) + n \tag{6.11}$$

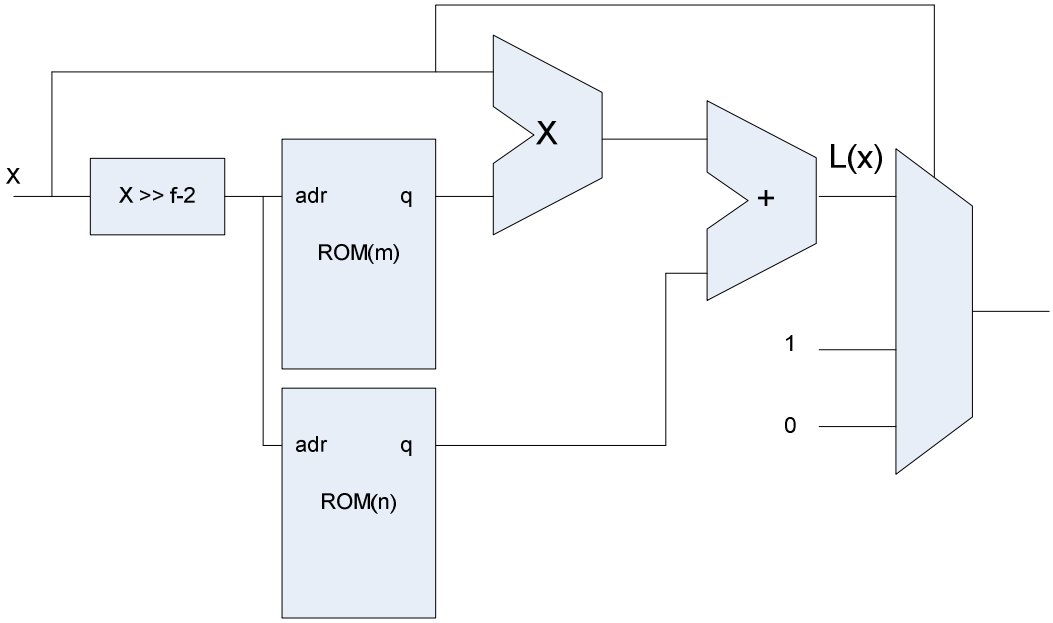


Şekil 6.12 Logsig fonksiyonunun parçalı doğrusal yaklaşım yöntemiyle 80 eşit parçaya bölünmüş hali

Parçalı doğrusal yaklaşım metodundaki matematiksel Eşitlik 6.12 ve 6.13 deki gibi yapılmaktadır. Bu işlemlerin donanımsal benzetimleri Şekil 6.13 de gösterildiği gibidir.

$$logsig(x) = \begin{cases} 1\,, x \geq 10 \\ 0\,, x \leq 10 \\ L(x), diğer \end{cases} \tag{6.12}$$

$$L(x) = ROM_m\big(floor(x * 4 + 1)\big) * x + \ ROM_n\big(floor(x * 4 + 1)\big) \tag{6.13}$$



Şekil 6.13 Logsig fonksiyonunun donanımsal benzetimi

6.5 Bölüm Çıktıları

Eğitilmek istenen YSA modülü ABC algoritmasına hedef fonksiyonu olarak eklenmiştir. ABC algoritması ile bulunan çözüm verktörleri YSA modülüne YSA ağırlıkları olarak alınıp ROM yapısında saklanan 400 adetlik eğitim bilgi seti ile Eşitlik 6.14 e göre bir maliyet değeri çıkartılmıştır.

$$f_i = (\sum_{k=1}^{400}(f_{YSA}(x_k) - f_{yd}(k))^2)/400 \quad (6.14)$$

Eşitlikte gösterilen f_i değeri ABC algoritması için maliyet değeri olup ABC algoritması bu değeri minimize etmeye çalışmaktadır. $f_{YSA}(x_k)$değeri k. Öznitelik setininden YSA nın aldığı cevaptır. $f_{yd}(k)$ ise k. Öznitelik setinin plaka olup olmadığını gösteren değerdir. Öz nitelik seti eğer plaka bölgesiyse $f_{yd}(k)$ cevabi 1 olacaktır ve $f_{YSA}(x_k)$ nında 1 e çok yakın bir değer olması istenir. Plaka olmayan bolgelerde ise $f_{yd}(k)$ cevabi 0 olacaktır ve $f_{YSA}(x_k)$ nında 0 a çok yakın bir değer olması istenir.

Eğitim işlemi bittikten sonra ABC algoritmasının bulduğu çözüm vektöründeki ağırlık değerleri YSA modülüne giriş olarak verilmiş ve test için oluşturulan 100 adet öznitelik bilgi seti ile eğitilen YSA'nın başarısı test edilmiştir. Test işleminin sonucu plaka

bölgesi veya plaka bölgesi olmayan bölgelerin doğru veya yanlış bulunma sayıları Çizelge 6.1 de verilmiştir.

Çizelge 6.1 Plaka bölgesi veya plaka bölgesi olmayan bölgelerin doğru veya yanlış bulunma sonuçları

Bulunan sonuçlar	Plaka bölgesi	Plaka bölgesi olmayan
Plaka bölgesi olarak bulunan	50	3
Plaka bölgesi değildir diye bulunan	0	47

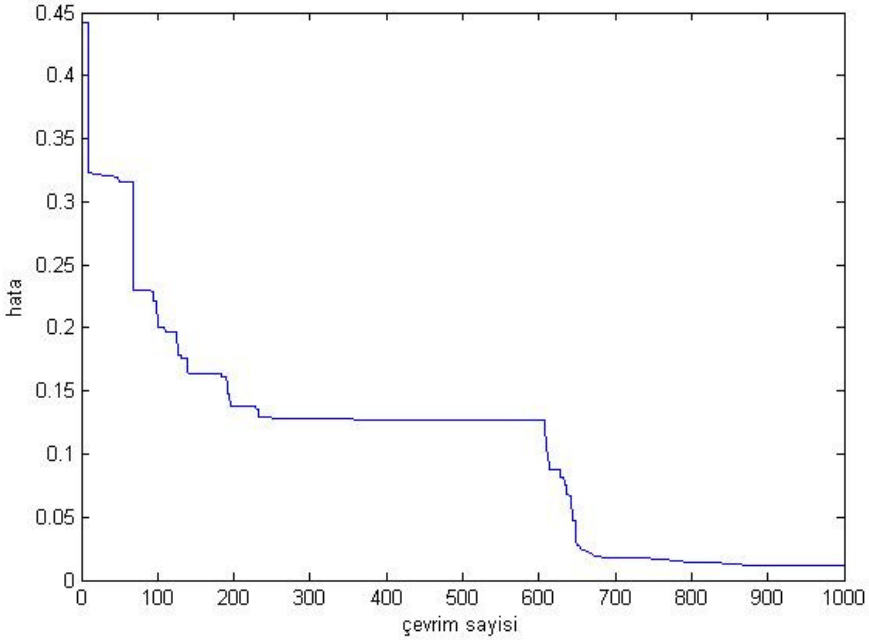
Yapay sinir ağı eğitiminde kullanılan ABC algoritmasına ait parametre değerleri ve YSA'nın eğitim ve test sonuçlar Çizelge 6.2'de verilmiştir.

Çizelge 6.2 FPGA üzerinde eğitilmiş YSA doğruluk hız ve kapladığı alan sonuçları

Platform	Arama uzayı aralığı	Yüzde Eğitim başarısı	Yüzde test başarısı	Hız(sn)	SN	maxCycle	limit	register	logic	memory
FPGA	40-(-40)	97	97	3,57	5	1000	100	3608	6721	240
PC (Matlab)	40-(-40)	98	97	1184	5	1000	100			

Matlab üzerinde eğitim sırasında her bir çevrim sonucu . ortalama hata değeri Şekil 6.14 de gösterildiği gibidir. Matlab üzerinde eğitim yaparken AMD Turion X2 Ultra Dual Core Mobile ZM-86 işlemcili ve 4 GB DDR2 RAM'e sahip 64 bit işletim sistemli PC kullanılmıştır.

Çizelgeden de görüldüğü üzere ABC algoritması ile YSA eğitim test sonuçları %97 dir. Donanım üzerindeki YSA eğitim süresi PC tabanlı uygulamasına göre 331 kat daha hızlıdır. Bunun sonucu olarak bu tür uygulamaların donanım üzerinde yapılması gerçek zamanlı sistemlerin tasarlanması için son derece önemlidir.



Şekil 6.14 Matlab üzerindeki YSA eğitimi

Şekil 6.14 de ABC algoritmanın her çevrimde bulduğu YSA eğitim hata sonuçlarını göstermektedir ve çevrim sayısı arttıkça hata küçülmektedir. İlk çevrimde hata oranı %45 iken, 1000 çevrim sonraki eğitimde hata oranı %2-3 e düşmektedir.

BÖLÜM VII

SONUÇ

Yarı iletken teknolojilerindeki gelişmelere paralel olarak evrimsel esnek hesaplama yöntemlerinin FPGA gibi paralel işlem yapmaya elverişli donanımlar üzerinde uyarlanması oldukça popüler bir araştırma alanı olmuştur. Bunun en önemli sebepleri donanımsal platformlardaki gelişmeler ve gerçek zamanlı uygulamalara duyulan ihtiyaçlardır. Bu tez çalışması kapsamında Yapay Arı Kolonisi algoritması (ABC) ilk kez FPGA üzerinde uyarlanmış ve yapının performansı yazılım emsali ile kıyaslanmaştır. ABC algoritmasının donanıma uyarlanması ile ilgili bir çalışma henüz literatürde mevcut değildir. Bu tez çalışması literatürdeki bu boşluğu doldurmaktadır. Tez kapsamında VHDL kullanılarak geliştirilen donanımsal mimari literatürden seçilen ve zor problemler olarak nitelendirilen bazı fonksiyonları optimize etmek için kullanılmıştır. Elde edilen sonuçlar incelendiğinde FPGA üzerinde koşturulan donanımsal mimarinin yazılım eşdeğerine göre çok daha hızlı çalıştığı ve üretilen sonuçların yaklaşık aynı doğrulukta olduğu gözlemlenmiştir. Yine tez kapsamında donanımsal olarak geliştirilen ABC algoritması bir YSA mimarisinin eğitiminde kullanılmıştır. Eğitim için seçilen YSA aday plaka bölgelerinden çıkarılan öznitelikleri kullanarak bölgenin plaka olup olmadığına karar vermektedir. FPGA üzerinde eğitilen YSA'nın eğitim setinden farklı 100 adet aday plaka bölgesi ile test edildiğinde başarı oranının %97'lere ulaştığı ve FPGA üzerinde yapılan eğitim ve test süresinin yazılım eşdeğerine göre 330 kat daha hızlı olduğu gözlemlenmiştir.

KAYNAKLAR

[1] Karaboga, D., An Idea Based On Honey Bee Swarm For Numerical Optimization, Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.

[2] Basturk, B. and Karaboga, D., An Artificial Bee Colony (ABC) Algorithm for Numeric function Optimization, IEEE Swarm Intelligence Symposium, Indianapolis, Indiana, USA, May 12-14, 2006.

[3] Karaboga, D. and Başturk, A., A powerful and Efficient Algorithm for Numerical Function Optimization: Artificial Bee Colony (ABC) Algorithm, Journal of Global Optimization, 39, 3, 459-171, 2007.

[4] Karaboga, D. and Basturk, B., On the performaıce of artificial bee colony (abc) algorithm, Applied Soft Computing, 8(1), 687-697, 2008.

[5] Karaboga. D. and Akay. B., Effect of region scaling on the initialization of particle swarm optimization differential evolution and artificial bee colony algorithms on multimodal high dimensional problems, International Conference on Multivariate Statistical Modelling and High Dimensional Data Mining, Kayseri, TURKEY, Jıne 19-23 2008.

[6] Karaboga, D. and Basturk, B., Advances in Soft Computing: Foundations of Fuzzy Logic and Soft Computing, LNCS, , chapter Artificial Bee Colony (ABC) Optimization Algorithm for Solving Constrained Optimization Problems, Springer-Verlag, 4529/2007, 789-798, 2007.

[7] Karaboga, D. and Akay, B., An artificial bee colony (abc) algorithm on training artificial neural networks, 15th IEEE Signal Processing and Communications Applications, SIU 2007, 1-4, Eskişehir, Türkiye, Jıne 2007.

[8] Karaboga, D., et al., Modeling Decisions for Artificial Intelligence, LNCS, volume 4617/2007, chapter Artificial Bee Colony (ABC) Optimization Algorithm for Training Feed-Forward Neural Networks. 318-329, Springer-Verlag, 2007.

[9] Karaboga, D., et al., Training neural Networks with abc optimization algorithm on medical pattern classification, International Conference on Multivariate Statistical Modeling and High Dimensional Data Mining, Kayseri, TURKEY, June 19-23 2008.

[10] Ozturk, C. and Karaboga, D., Classification by neural networks and clustering with artificial bee colony (abc) algorithm, 6th International Symposium on Intelligent and Manufacturin Systems Features, Strategies and Innovation, Sakarya, Türkiye, October 14-17 2008.

[11] Karaboga, D. and Ozturk C., Neural Networks Training by Artificial Bee Colony Algorithm on Pattern Classification, Neural Network World, 19 (3), 279-292, 2009.

[12] Akay, B., Nümerik optimizasyon problemlerinde yapay arı kolonisi(Artificial bee colony) algoritmasının performans analizi, Doktora Tezi, Erciyes Ü., Fen bilimleri enstitüsü, Kayseri, 2009.

[13] Juang, J., and Lin B., Hybrid Intelligent Aircraft Landing Controller and Its Hardware Implementation, ICNC2006, Part II, LNCS 4222, pp. 972-981, 2006

[14] Farmahini-Farahani, A., Fakhraie, S. and Safari, S., SOPC Based Architecture for Discrete Particle Swarn Optimization, ICECS 2007, 1003-1006, Morocco, 11-14 Dec. 2007

[15] Drechsler, R. and Drechsler, N., GAME-HDL: Implementation of Evolutionary Algorithms using Hardware Description Languages , Application of Evolutionary Computing, Lecture Notes in Computer Science, 2611, 378-387, 2003

[16] Avci G., Kosten M.M., Altun H., Karakaya F. and Cavuslu M.A., Implementation of an Hybrid Approach on FPGA for License Plate Detection Using Genetic Algorithm

and Neural Networks International Symposium on Inovations in Intelligent SysTems and Applications, 392-396, June 29-July 1, 2009, Trabzon.

[17] llaire, F.C.J., Tarbouchi, M., Labonte, G. and Fusina, G., FPGA Implementation of Genetic Algorithm for UAV Real-Time Path Planing, J. Intell Robot Syst, 54, 495-510, 2009

[18] Dorigo, M., Maniezzo, V. and Colorni, A., Positive Feedback as a Search Strategy, Tech. Rep. 91-016, Politechnico di Milano, Italy , 1991

[19] Scheuermann, B., So, K. and Guntsch, M., Middendorf, M., Diesel, O., Elgindy, H., Schmek, H., FPGA Implementation of population-based ant colony optimization, Applied Soft Computing, 4, 303-322 , 2004

[20] Juang, C.F., Lu, C.M. and Wang, C.Y., Ant Colony Optimization Algorithm for Fuzzy Controller Design and its FPGA Implementation, IEEE Trans. on Industrial Electronics, 55, 3, 1453-1462, 2008.

[21] Lin, C.J. and Tsai, H.M., FPGA implementation of a wavelet neural network with particle swarn optimization learning, Mathematical and Computer Modelling, 47, 982-996, 2008

[22] Bonabeau, E., et al.. Swarm intelligence: from natural to artiticial systems, Oxford University Press. Inc., New York, NY, USA, 1999, ISBN 0195131592, URL http://portal.acm.org/citation.cfm?id=328320.

[23] Beekman, M., et al., What makes a honeybee scout?. Behavioral Ecology and Sociobiology, 61, 985 995, 2007 URL http: //dx.Doi.org/10.1007/s00265-006-0331-9.

[24] Domhaus. A., el al., Task selection in honeybees - experiments using multi-agent simulation. In Proc of GWAL'98, September 18-19 1998.

[25] Von Frisch, K., The Dancing Bees: An Account of the Life and Senses of Honey Bee, Harcourt, Brace, 1953.

[26] Von Frisch, K. and Lindauer, M., The "language" and orientation of the honey bee, Annu. Rev. Entomol, 1, 45-58. 1956.

[27] Seeley, T., Honeybee ecology: A study of adaptation in social life, Princeton University Press, Princeton, 1985.

[28] Tereshko, V., Reaction-diffusion model of a honeybee colony's foraging behaviour, PPSN VI: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature, 807-816, Springer-Verlag, London, UK, 2000, ISBN 3-540-41056-2.

[29] Hamdan, K., How do bees raake honey, 2008, bee Research Unit, National Center for Agriculture Research and Technology Transfer, bee. (NCARTT), http://www.jordanbru.info/how do Bees make hony.htm.

[30] Elmas, Ç., Yapay Zeka Uygulamaları, Seçkin Yayıncılık, İstanbul 2007.