T.C.
ERCİYES ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

PARALEL PROGRAMLAMA TEKNİKLERİNİN GELİŞİME DAYALI ALGORİTMALAR ÜZERİNDEKİ ETKİNLİK ANALİZİ

Tezi Hazırlayan
Rüştü AKAY

Tezi Yöneten
Doç. Dr. Alper BAŞTÜRK

Bilgisayar Mühendisliği Anabilim Dalı
Doktora Tezi

Ağustos 2014
KAYSERİ

T.C.
ERCİYES ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

PARALEL PROGRAMLAMA TEKNİKLERİNİN GELİŞİME DAYALI ALGORİTMALAR ÜZERİNDEKİ ETKİNLİK ANALİZİ

Tezi Hazırlayan
Rüştü AKAY

Tezi Yöneten
Doç. Dr. Alper BAŞTÜRK

Bilgisayar Mühendisliği Anabilim Dalı
Doktora Tezi

Bu çalışma Erciyes Üniversitesi Bilimsel Araştırma Projeleri Koordinasyon Birimi tarafından FBD-10-3117 kodlu proje ile desteklenmiştir

Ağustos 2014
KAYSERİ

Doç. Dr. Alper BAŞTÜRK danışmanlığında **Rüştü AKAY** tarafından hazırlanan **"Paralel Programlama Tekniklerinin Gelişime Dayalı Algoritmalar Üzerindeki Etkinlik Analizi"** adlı bu çalışma, jürimiz tarafından Erciyes Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalında **Doktora** tezi olarak kabul edilmiştir.

11.08.2014

JÜRİ :

Başkan : Prof. Dr. Halûk GÜMÜŞKAYA

Üye : Prof. Dr. Hamza EROL

Üye : Prof. Dr. Coşkun ÖZKAN

Üye : Prof. Dr. Mehmet Emin YÜKSEL

Üye : Doç. Dr. Alper BAŞTÜRK

ONAY :

Bu tezin kabülü Enstitü Yönetim Kurulunun ..... tarih ve ..... sayılı kararı ile onaylanmıştır.

.... / .... / .....

Prof. Dr. Kâzım KEŞLİOĞLU

Enstitü Müdürü

Doç. Dr. Alper BAŞTÜRK danışmanlığında **Rüştü AKAY** tarafından hazırlanan **"Paralel Programlama Tekniklerinin Gelişime Dayalı Algoritmalar Üzerindeki Etkinlik Analizi"** adlı bu çalışma, jürimiz tarafından Erciyes Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalında **Doktora** tezi olarak kabul edilmiştir.

11.08.2014

JÜRİ :

Başkan : Prof. Dr. Halûk GÜMÜŞKAYA

Üye : Prof. Dr. Hamza EROL

Üye : Prof. Dr. Coşkun ÖZKAN

Üye : Prof. Dr. Mehmet Emin YÜKSEL

Üye : Doç. Dr. Alper BAŞTÜRK

ONAY :

Bu tezin kabülü Enstitü Yönetim Kurulunun ..... tarih ve ..... sayılı kararı ile onaylanmıştır.

.... / .... / .....

Prof. Dr. Kâzım KEŞLİOĞLU

Enstitü Müdürü

Doç. Dr. Alper BAŞTÜRK danışmanlığında **Rüştü AKAY** tarafından hazırlanan **"Paralel Programlama Tekniklerinin Gelişime Dayalı Algoritmalar Üzerindeki Etkinlik Analizi"** adlı bu çalışma, jürimiz tarafından Erciyes Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalında **Doktora** tezi olarak kabul edilmiştir.

11.08.2014

JÜRİ :

Başkan : Prof. Dr. Halûk GÜMÜŞKAYA

Üye : Prof. Dr. Hamza EROL

Üye : Prof. Dr. Coşkun ÖZKAN

Üye : Prof. Dr. Mehmet Emin YÜKSEL

Üye : Doç. Dr. Alper BAŞTÜRK

ONAY :

Bu tezin kabülü Enstitü Yönetim Kurulunun ..... tarih ve ..... sayılı kararı ile onaylanmıştır.

.... / .... / .....

Prof. Dr. Kâzım KEŞLİOĞLU

Enstitü Müdürü

TEŞEKKÜR

"Paralel Programlama Tekniklerinin Gelişime Dayalı Algoritmalar Üzerindeki Etkinlik Analizi" konulu tez çalışmamın yürütülmesinde, zamanını, kıymetli bilgi ve yardımlarını esirgemeyen saygıdeğer hocam Doç. Dr. Alper BAŞTÜRK'e, optimizasyon konusunu bizlere sevdiren saygıdeğer hocam Prof. Dr. Derviş KARABOĞA'ya ve tez izleme komitemde yer alan hocalarım Prof. Dr. Halûk Gümüşkaya'ya, Prof. Dr. Hamza Erol'a, Prof. Dr. Coşkun Özkan'a ve Prof. Dr. Mehmet Emin YÜKSEL'e saygı ve teşekkürlerimi sunarım.

Hayatım boyunca verdikleri tüm desteklerden ötürü haklarını ödeyemeceğim aileme, desteğinden dolayı eşim Bahriye Akay'a, sağladıkları motivasyondan ötürü oğlum Ali Batu'ya ve kızım Hatice Betül'e teşekkür ederim.

Erciyes Üniversitesi Bilimsel Araştırma Projeleri Koordinasyon Birimi'ne FBD-10-3117 kodlu proje desteğinden dolayı teşekkürlerimi sunarım.

PARALEL PROGRAMLAMA TEKNİKLERİNİN GELİŞİME DAYALI ALGORİTMALAR ÜZERİNDEKİ ETKİNLİK ANALİZİ

Rüştü AKAY
Erciyes Üniversitesi, Fen Bilimleri Enstitüsü
Doktora Tezi, Ağustos 2014
Tez Danışmanı : Doç. Dr. Alper BAŞTÜRK

ÖZET

Son yıllarda yaşanan teknolojik gelişmeler tasarım problemlerinin daha da zorlaşmasını beraberinde getirmiş ve hesaplama ihtiyaçlarını artırmıştır. Problemlerin zorlaşması ve artan hesaplama ihtiyacının karşılanması, çözümlerde kullanılan algoritmaların performanslarını arttırıcı yeni yaklaşımlar önerilmesine ve paralel hesaplama sistemlerinin etkin bir şekilde kullanılmasına olan ilgiyi arttırmaktadır.

Bu amaçla tez kapsamında, temel versiyonları asenkron yapıda olan bazı algoritmaların senkron modelleri önerilerek performanslarını arttırıcı yeni yaklaşımlar geliştirilmiştir. Aynı zamanda, algoritmaların performanslarının kontrol parametrelerine olan bağımlıklarını önlemek ve daha kararlı yapılar oluşturmak için yeni portföy stratejileri önerilmiştir. Önerilen yeni modellerin farklı paralel gerçekleştirimleri ile, değişik özelliklere sahip zorluk derecesi yüksek test problemlerinin çözümü gerçekleştirilmiş ve detaylı başarım analizleri yapılmıştır. Bilinen test problemlerinin yanı sıra, gerçek dünya problemlerinden yapay sinir ağları eğitimi gerçekleştirilmiştir. Bunlara ilave olarak bazı algoritmaların ayrık modellerinin paralel gerçekleştirimi yapılarak etkin komşu üretme mekanizmaları entegre edilmiştir. Bu gerçekleştirimler ayrık bir problem türü olan gezgin satıcı problemlerinin çözümünde kullanılmıştır. Elde edilen sonuçlardan, önerilen modellerin problemlerin çözüm kalitesini arttırdığı, algoritmaları daha kararlı hale getirdiği ve paralel hesaplama sistemlerinin kullanımı ile de çözüm sürelerinin önemli oranda kısalabileceği gözlemlenmiştir. Bu çalışma ile paralel hesaplama sistemleri üzerinde çalışan algoritmalar ile geniş ve kapsamlı problemlerin etkin bir şekilde çözülebileceği gösterilmiştir.

Anahtar Kelimeler: Gelişime dayalı algoritmalar, paralel programlama

EFFICIENCY ANALYSIS OF PARALLEL PROGRAMMING TECHNIQUES ON EVOLUTIONARY ALGORITHMS

Rüştü AKAY
Erciyes University, Graduate School of Natural and Applied Sciences
Ph.D. Thesis, 2014
Thesis Supervisor: Assoc. Prof. Dr. Alper BAŞTÜRK

ABSTRACT

Technological advances in recent years have made design problems more difficult and have increased the computing needs required to solve them. The difficulty of problems and increased source needs have led to the proposal of parallel implementations of some commonly used problem solving techniques and to increased interest in efficient parallel computation systems.

For this purpose, synchronous models of the algorithms, the basic versions of which have asynchronous structures, were proposed in order to achieve faster parallel models. Moreover, to prevent performance from being affected by the control parameters and to make more stable structures, new portfolio strategies were proposed. Difficult problems with different characteristics and with varying difficulty levels were solved by the proposed approaches and comprehensive comparisons were presented in the experiments for performance analysis. In addition to well-known benchmark problems, some real-world problems were solved by using artificial neural networks and trained by using the proposed parallel models. Moreover, the parallel models of the algorithms were implemented for combinatorial type problems and efficient local search methods were integrated into the algorithms. The traveling salesman problem was used in the experiments carried out for combinatorial algorithms. It was observed that the proposed models improved solution quality and algorithm stability; they also decreased the running times significantly. It is shown that evolutionary algorithms running on parallel computing systems can be employed to solve large and complex problems effectively.

Keywords: Evolutionary algorithms, parallel programming

Doç. Dr. Alper BAŞTÜRK danışmanlığında **Rüştü AKAY** tarafından hazırlanan **"Paralel Programlama Tekniklerinin Gelişime Dayalı Algoritmalar Üzerindeki Etkinlik Analizi"** adlı bu çalışma, jürimiz tarafından Erciyes Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalında **Doktora** tezi olarak kabul edilmiştir.

11.08.2014

JÜRİ :

Başkan : Prof. Dr. Halûk GÜMÜŞKAYA

Üye : Prof. Dr. Hamza EROL

Üye : Prof. Dr. Coşkun ÖZKAN

Üye : Prof. Dr. Mehmet Emin YÜKSEL

Üye : Doç. Dr. Alper BAŞTÜRK

ONAY :

Bu tezin kabülü Enstitü Yönetim Kurulunun ..... tarih ve ..... sayılı kararı ile onaylanmıştır.

.... / .... / .....

Prof. Dr. Kâzım KEŞLİOĞLU

Enstitü Müdürü

İÇİNDEKİLER

4. BÖLÜM

5. BÖLÜM

TABLOLAR LİSTESİ

ŞEKİLLER LİSTESİ

GİRİŞ

Son yıllarda yaşanan teknolojik gelişmeler çok bileşenli ve karmaşık yapılardan oluşan tasarım problemlerinin daha da zorlaşmasını beraberinde getirmiş ve hesaplama ihtiyaçlarını artırmıştır. Genel olarak problemlerin kesin çözümlerini bulmak için geliştirilmiş klasik yöntemler mevcuttur, fakat problemler zorlaştıkça klasik yöntemler hem çözümü güçleştirmekte hem de yüklü bir işlem maliyeti gerektirmektedir. Ayrıca analitik olarak modellenemeyen birçok problemin çözümünde de klasik yöntemlerin uygulanabilirliği mümkün olamamaktadır. Bu tür büyük boyutlu ve zor problemlerin çözümünde klasik yöntemlerin genellikle yetersiz kalması, araştırmacıları esnek hesaplama teknikleri gibi yeni ve etkili çözüm yöntemleri geliştirmeye yöneltmektedir.

Esnek hesaplama (Soft Computing), analitik olarak çözümü öngörülemeyen problemlerin çözümleri için geliştirilmiş tekniklere verilen genel bir terimdir. Bu terim genel olarak bilgisayar bilimlerinde kullanılır ve bulanık mantık (Fuzzy Logic), yapay sinir ağları (Artificial Neural Networks) ve evrimsel hesaplama (Evolutionary Computation) gibi teknikleri kapsamaktadır. Bu teknikler sahip oldukları modelleme, akıl yürütme, yorumlama ve tahmin edebilme gibi özellikleri sayesinde karmaşık sorunlara karmaşık olmayan çözümler sunabilmektedirler. Esnek hesaplama teknikleri, sahip oldukları bu özellikler sayesinde birçok alanda yaygın olarak kullanılmaktadır.

Bulanık mantık kavramı, belirsizliklerin anlatımı ve belirsizliklerle çalışılabilmesini sağlayan matematiksel düzen olarak tanımlanmaktadır. Yapay sinir ağları, insan beyninin çalışma mantığından esinlenilerek insanların sahip olduğu düşünebilme ve öğrenebilme yeteneklerinin simüle edilmeye çalışılmasıdır. Evrimsel hesaplama ise biyolojik evrim ilkelerine dayalı problem çözme tekniklerine verilen ortak

isimdir. Sezgisel algoritmaları da içeren bu teknikler çoğunlukla karmaşık problemlerin çözümlerinde kullanılmaktadır. Sezgisel algoritmalar herhangi bir amacı gerçekleştirmek için çeşitli alternatif hareketlerden etkili olanları seçme yöntemleridir. Genel olarak doğada var olan olayları modellerler, rassaldırlar ve problemlerin kesin çözümünü elde edeceklerini garanti edemezler.

Bu yöntemler klasik yöntemlere göre daha başarılı sonuçlar sağlamasına rağmen, büyük boyutlu problemlerin çözümünün genellikle kabul edilebilir sürelerin ötesinde hesaplama zamanı gerektirmesi ayrı bir problem olarak ortaya çıkmaktadır. Bu problemin aşılması amacıyla da uygulanan yaklaşım genellikle paralel hesaplama sistemlerinin kullanılmasıdır.

Esnek hesaplama yöntemlerinin paralel programlama teknikleri kullanılarak paralelleştirilmesi ile problemler parçalara bölünmekte ve parçaların çok işlemcili donanım mimarileri üzerinde eş zamanlı olarak işletilmesi sağlanmaktadır. Bu sayede büyük boyutlu problemlerin kabul edilebilir süreler içerisinde çözümleri gerçekleştirilebilmektedir.

1. BÖLÜM

GENEL BİLGİLER

1.1. Giriş

Gerçek dünya problemlerinin genellikle zor, karmaşık ve zaman alıcı olması, klasik yöntemlerle çözülebilmelerini güçleştirmektedir. Bu tür problemlerin çözümlerinde sezgisel algoritmalarının başarılı sonuçlar üretmesi konuya olan ilgiyi her geçen gün arttırmaktadır. Araştırmacılar farklı doğal olayları modelleyen yeni sezgisel algoritmalar önermekte ve başarımlarını farklı optimizasyon problemleri üzerinde incelemektedirler.

Sezgisel algoritmalarının popüler olanlarından bazıları şunlardır: genetik algoritmalar (Genetic Algorithms, GA) [1], tabu araştırma (Tabu Search, TS) [2, 3], benzetilmiş tavlama (Simulated Annealing, SA) [4, 5], karınca kolonisi optimizasyonu (Ant Colony Optimization, ACO) [6], yapay arı koloni (Artificial Bee Colony, ABC) [7], diferansiyel gelişim (Differential Evolution, DE) [8] ve parçacık sürü optimizasyonu (Particle Swarm Optimization, PSO) [9]. Bu algoritmaların karmaşık problem çözümlerinde başarılı olması, araştırmacıları doğal olayları modelleyen farklı algoritmalar önermeye teşvik etmektedir. Bu amaçla önerilen güncel algoritmalara, guguk kuşu arama (Cuckoo Search, CS) [10], ateş böceği (Firefly, FF) [11], gravitasyonel arama (Gravitational Search, GS) [12] ve öğretme ve öğrenme temelli Optimizasyon (Teaching-Learning-Based Optimization, TLBO) [13, 14] örnek olarak verilebilir.

Farklı tipteki optimizasyon problemlerine etkili çözüm arayışları, araştırmacıların yeni algoritmalar önermelerinin yanında mevcut algoritmalara ek bazı iyileştirme

stratejileri geliştirmelerini de teşvik etmektedir. Bu çalışmalar da algoritmaların farklı problem türlerinde göstermiş oldukları başarımlar detaylı analiz edilerek, eksik yönleri giderilmek amacı ile farklı iyileştirme stratejileri önerilmektedir. ABC algoritmasının performansını artırmak için önerilen yeni yaklaşımlar [15–22], DE algoritmasının performansını artırmak için önerilen yeni yaklaşımlar [23, 24], PSO algoritmasının performansını artırmak için önerilen yeni yaklaşımlar [25–27] ve TLBO algoritmasının performansını artırmak için önerilen yeni yaklaşımlar [28, 29] bunlardan sadece bazılarıdır. Algoritmaların performanslarını iyileştirmek için entegre edilen bu mekanizmalar, algoritmaların modellediği doğal olaylardan esinlenmeyebilir. Bu mekanizmalar genellikle algoritmaların sahip oldukları dezavantajları gidermek amacı ile yapılan ufak modifikasyonlardır.

Sezgisel algoritmalar genellikle olasılık tabanlıdır ve bazı kontrol parametreleri içerirler. Kontrol parametrelerinin değerleri, algoritmaların performanslarını önemli derecede etkilemektedir. Algoritmaların sahip oldukları kontrol parametreleri için önerilen bazı değerler olmasına rağmen [8, 30–34], kontrol parametre değerleri ve algoritma performansları arasındaki etkileşim hala tam olarak anlaşılmış değildir. Bunun temel nedeni farklı problem türlerinde farklı değerlerin daha başarılı olması ve kontrol parametrelerinin hiç bir sabit ayarının olmamasıdır. Bu sebeple çözülmek istenen probleme göre kontrol parametre değerlerinin ayrıca ayarlanması gerekir. Bunun yanında kontrol parametre sayısının fazlalığı, ayarlanmaları için ihtiyaç duyulan deney sayısının da fazla olması demektir. Dolayısı ile bu işlemler yüksek hesaplama maliyetlerini ve uzun çalışma sürelerini beraberinde getirmektedir.

Araştırmacılar algoritmaların performanslarını iyileştirmek için yaptıkları çalışmaların yanında, kontrol parametrelerinin kendi kendine uyarlanabilir olduğu bazı çalışmalar da yapmaktadırlar [11, 31, 35–54]. Bu çalışmalarda problem çözüm kaliteleri üst düzeye çıkarılırken, algoritmaların performanslarının kontrol parametrelerine olan bağımlılığı önlenmiş olmaktadır. Aynı zamanda kullanıcıların çalışmalarında optimizasyon problemlerinin özellikleri ile kontrol parametre değerleri arasındaki ilişkileri bilmelerine gerek kalmamaktadır. Ayrıca uyarlanabilir parametreler iyi tasarlanırsa algoritmaların yakınsama performanslarıda iyileşebilmektedir.

Oldukça geniş bir araştırma alanını kapsayan sezgisel algoritmaların geliştirilmiş modelleri ve farklı optimizasyon problemleri çözümlerinde kullanımları ile ilgili detaylı bilgiler bazı araştırma makalelerinde verilmektedir [55–63].

Gerçekleştirilen birçok uygulama sezgisel algoritmalar ile elde edilen çözümlerin klasik yöntemler ile sağlanan çözümlere göre üstünlükler taşıdığını göstermektedir. Ancak problem boyutları büyüdükçe, sezgisel algoritmalar da çözüm için daha fazla zamana ihtiyaç duymaktadır.

Günümüzde hızla gelişen bilgisayar teknolojileri bile büyük boyutlu problemlerin çözümlerini kabul edilebilir sürelerde gerçekleştirememektedir. Bunun sebebi ise işlemci hızının günümüz şartlarında neredeyse fiziksel limitlere ulaşmış olmasıdır. Bu zorluğun üstesinden gelmek için işlemci hızını arttırmak yerine işlemci sayısının arttırıldığı paralel mimariler tercih edilmektedir. Ancak, araştırmacıların bu paralel mimarilerin avantajlarından faydalanabilmesi için de geliştirmiş oldukları teknikleri paralel çalışabilecek şekilde uyarlamaları gerekmektedir.

Bu bağlamda yapılan çalışmalar esnek hesaplama yöntemlerinin paralel programlama teknikleri kullanılarak paralelleştirimesi ve paralel hesaplama sistemleri üzerinde çalıştırılmasıdır. Paralel programlama tekniklerinin arkasındaki temel fikir, problemin parçalara bölünmesi ve her bir parçanın aynı anda çözümünün gerçekleştirilmesidir. Paralel hesaplama sistemleri ise, bir problemin parçalarının birkaç işlemci üzerinde aynı anda çözümlenebilmesine olanak sağlayan bilgisayar mimarilerdir. Bu sistemler çok çekirdekli işlemciler, ortak ağa bağlı birçok bilgisayar, özel bir donanım veya bunların herhangi bir kombinasyonu olabilir. Problemlerin parçalara bölünerek paralel hesaplama sistemleri üzerinde çözülme yaklaşımı çok farklı şekillerde esnek hesaplama yöntemlerine uyarlanabilir. Tez kapsamında genel olarak esnek hesaplama yöntemlerinden popülasyon tabanlı sezgisel algoritmalar üzerinde çalışıldığı için, bu algoritmaların paralel uyarlamaları üzerinde durulmuştur. Popülasyon tabanlı sezgisel algoritmaların paralel modellerinin sınıflandırılması, üç temel model (Master-Slave, Fine-Grained, Coarse-Grained) ve bu modellerin değişik kombinasyonlarını içeren hybrid modeller şeklindedir. Bu modellerden bazıları tek bir popülasyondan oluşurken, bazıları

bölünmüş kısmen etkileşimli alt popülasyonlardan oluşur. Bazı modeller büyük ölçekli paralel makineler için uygunken, diğerleri az sayıda güçlü işlemciler için daha uygundur. Bu modeller ile ilgili temel ve teorik bilgiler ayrı bir bölümde detaylandırılmıştır.

1.2. Literatür Taraması

Paralel hesaplama sistemlerine olan ilginin son dönemlerde artmasına rağmen, paralel makineler fikri çok daha eskilere dayanmaktadır. Örneğin, 1950'lerin sonlarında John Holland çok sayıda programı eşzamanlı çalıştıran paralel bilgisayarlar için bir mimari önermiştir [64, 65]. Holland'ın önerdiği bu mimari birbirine bağlı nispeten basit bilgisayar donanımlarından oluşmaktaydı. Bu makineler üzerinde çalışan programlar özellikle donanım problemlerine uyum sağlayarak kendilerini değiştirebiliyorlardı. Ancak bilgisayar yazılımları paralel programlama için çok esnek olmadığı için uygulama geliştirmek oldukça zordu. Bu bilgisayarlar Holland tarafından doğal popülasyonların evrim simülasyonunu gerçekleştirmek için kullanılmıştır.

1.2.1. İlk çalışmalar

Holland tarafından önerilen GA ilk sezgisel algoritma olması sebebi ile ilk paralel uygulamalarda GA ile gerçekleştirilmiştir. 1976 yılında Bethke tarafından standart GA'ya uyarlanan master-slave paralelleştirme modeli bu alanda ilk uygulamadır [66]. Çalışmada paralel GA'lar ile tek komut çok veri akışı (Single Instruction Multiple Data, SIMD) mimarili bilgisayar sistemlerinde %100'e yakın bir verimlilik elde edilebileceği gösterilmiştir. Ayrıca bu çalışmada paralel verimliliği sınırlayan darboğazların tespit edilmesi için analizler yapılmıştır. İlk yapılan çalışmalardan bir diğeri Grefenstette tarafından GA'nın o zamanki mevcut bilgisayar mimarileri üzerinde nasıl paralelleştirilebileceği üzerinedir [67]. Grefenstette çalışmasında paralel GA için dört model önermiştir. Bunlardan ilk üçü master-slave modelinin varyasyonları dördüncüsü ise çok popülasyonlu paralel GA'dır. Birinci model de master işlemci popülasyon bilgilerinin hafızada tutulması, yeni nesil üretmek için bireylerin seçilmesi, çaprazlama ve mutasyon işlemlerinin gerçekleştirilmesinden

sorumludur. Bu modelde bireyler slave işlemcilere uygunluk değerlerinin hesaplanması için gönderilir. Slave işlemciler ise sadece bireylerin uygunluk değerlerini hesaplar ve master işlemciye gönderir. İkinci modelde birinci modele oldukça benzerdir, ancak nesiller arasında açık bir bölünme yoktur. Herhangi bir slave işlemci bireyin uygunluk değerini hesapladıktan sonra ana işlemciye gönderir ve başka bir bireyi alır. Dolayısı ile bu modelde sekronizasyon yoktur ve slave işlemciler farklı hızlarda çalışıyor olsalar bile maksimum fayda sağlanır. Üçüncü modelde ise popülasyon tüm slave işlemcilerin erişebileceği paylaşımlı bellekte tutulur. Slave işlemciler birbirlerinden bağımsız olarak genetik operatörler uygularlar. Dördüncü model, çok popülasyondan oluşan paralel GA'da herbir popülasyonun farklı bir işlemcide çalışması ve işlemcilerdeki en iyi bireylerin her iterasyonda diğer işlemcilere yayınlanmasıdır. Çok popülasyonlu paralel algoritmaların kullanımı ile ilgili ilk uygulamalardan bir diğeri Grosso tarafından gerçekleştirilmiştir [68]. İlerleyen zamanlarda bu çalışmaları daha sistematik çalışmalar izlemiştir. GA ile yapılan paralelleştime modelleri ve analizler diğer algoritmalar için de yapılmıştır ve yapılmaktadır. Tabu araştırma için [69], ısıl işlem algoritması için [70], evrimsel stratejilerin için [71], evrimsel programlama için [72], karınca kolonisi optimizasyonu için [73] ilk paralel çalışmalardan bazılarıdır.

1.2.2. Master-Slave Modeline Dayalı Çalışmalar

Fogarty ve Huang, simülasyonu aldukça fazla süreler alan kutup dengeleme uygulaması için master-slave paralel GA gerçekleştirmişlerdir [74]. Uygulamalarında paralel hesaplamalar için özel olarak hazırlanmış mikroişlemcilerden oluşan bir ağ kullanmışlardır. Çalışmada makul ölçüde hızlanma elde ederken, hızlanmaya engel olan iletişim yükünü yorumlamışlardır. Abramson ve Abela paylaşımlı bellek bilgisayarda (16 işlemcili Encore Multimax) paralel GA ile ders çizelgeleme uygulaması gerçekleştirmişlerdir [75]. Bu çalışmada sınırlı bir hızlanma elde etmişlerdir. Daha sonra Abramson ve arkadaşları dağıtık bellek bilgisayarda (128 işlemcili Fujitsu AP1000) ders çizelgeleme uygulamasını gerçekleştirmişlerdir [76]. Bu iki bilgisayar üzerinde 16 işlemci ile gerçekleştirilen uygulamalar karşılaştırıldığında elde edilen hızlanma hemen hemen aynı olmuştur. Ancak, daha

fazla işlemci ilave edildikçe, özellikle artan iletişim maliyeti sebebi ile verimlilik önemli ölçüde düşmüştür.

Venter ve Sobieszczanski-Sobieski senkron ve asenkron master-slave paralel PSO algoritmalarını gerçekleştirerek, başarımlarını incelemişlerdir. Çalışmada paralel verimlilik açısından asenkron paralel PSO algoritmasının senkron paralel PSO algoritmasına göre daha iyi olduğu görülmüştür [77]. Benzer bir çalışmada Scriven ve arkadaşları tarafından gerçekleştirilmiştir. Gerçekleştirdikleri senkron ve asenkron paralel PSO algoritmalarının performanslarını heterojen problemler üzerinde test etmişlerdir. Bu çalışmada da asenkron paralel PSO algoritmasının senkron paralel PSO algoritmasına göre yakınsama hızının daha yavaş, paralel verimliliğinin daha fazla olduğu vurgulanmıştır [78].

Narasimhan popülasyon boyutuna eşit sayıda işlemci kullanarak ABC algoritmasının ilk paralel uygulamasını gerçekleştirmiştir [79]. Basturk ve Akay ABC algoritmasının senkron modelini önermişler, senkron ABC algoritmasının master-slave paralel gerçekleştirimini yapmışlardır. Sonuçlardan senkron ve asenkron ABC algoritmasının performansları arasında anlamlı bir fark olmadığı, paralel senkron ABC algoritmasının fonksiyon hesaplamalarının maliyetli olduğu problemlerde oldukça avantajlı olacağını belirtmişlerdir [80].

Sonuç olarak, popülasyon tabanlı sezgisel algoritmalar için master-slave paralel model, gerçekleştirimi oldukça kolay, uygunluk değerlerinin hesaplama maliyetleri fazla olan problemler içinde oldukça etkili bir yöntem olduğu gösterilmiştir. Ayrıca, bu yöntem genel olarak algoritmaların arama davranışını değiştirmediği için de algoritmalar için geliştirilen tüm teorilerin doğrudan uygulanabileceği belirtilmektedir.

1.2.3. Fine-Grained Modeline Dayalı Çalışmalar

Manderick ve Spiessens 2D grid üzerinde fine-grained parallel GA gerçekleştirmişlerdir [81]. Çalışmada seçim ve yeni çözüm üretme işlemi yerel bir komşuluk sınırları içrisinde mümkün olmaktadır. Ayrıca yerel komşuluk sınırlarının genişlemesinin algoritmanın performansını etkilediği gözlemlenmiştir.

Çalışmada yerel komşuluk sınırları yeterince büyük olduğu taktirde, paralel GA'nın seri GA'ya eşdeğer olduğu söylenmiştir.

Sarma and De Jong yerel komşuluk sınırlarının boyutu ve şeklinin seçim mekanizması üzerindeki etkilerini analiz etmiştir. Çalışmada yerel komşuluk sınırlarının yarıçapının, bütün grid yarıçapına oranının modelin performasını etkileyen önemli bir parametre olduğu vurgulanmıştır [82].

1.2.4. Coarse-Grained Modeline Dayalı Çalışmalar

Coarse-grained paralelleştirme modeli popülasyon tabanlı sezgisel algoritmalar için performans kazançları açısından çok umut vericidir. Bunun yanında bu model algoritmaların seri eş değerlerine göre daha karmaşıktır. Özellikle alt popülasyonlar arası bireylerin geçişi çeşitli parametreler tarafından kontrol edilmektedir. Özetle bu parametreler, (a) alt popülasyonlar arasındaki bağlantıyı tanımlayan topoloji, (b) kaç bireyin göç edeceği ve (c) göç aralığı olarak tanımlanabilir. Bu parametrelerin belirlenmesi için 1980'lerin sonu ve 1990'ların başlarında ilk teorik ve deneysel çalışmalar yapılmaya başlamıştır.

Parametreleri analiz etmek için standart genetik algoritmanın çok popülasyonlu paralel modeli Tanese [83], Pettey ve arkadaşları [84, 85] tarafından yapılmıştır. Tanese çalışmasında göç topolojisi olarak dört boyutlu hiper küp topolojisi gerçekleştirmiştir. İşlemciler arası göç sabit aralıklarla küpün bir boyutu üzerinden yapılmıştır. Göç edecek birey, olasılığa dayalı olarak en iyi bireylerden seçilmiş ve gönderilen işlemcide en kötü bireyin yerini almıştır. Üç farklı deney yapılan çalışmada ilk olarak, göç sıklığı sabitlenerek farklı işlemci sayılarında paralel GA gerçekleştirilmiştir. Bu deneyde paralel GA ile seri GA aynı kalitede sonuçlar üretmiştir. İkinci deneyde ise algoritmanın keşif ve sömürü özelliğini dengelemek için her bir alt popülasyonda farklı mutasyon ve çaprazlama oranları kullanarak uygun değerleri bulmaya çalışmıştır. Üçüncü deneyde ise farklı göç sıklığı değerlerinin performansa etkisini incelemiştir. Sonuçlardan çok sık yada çok seyrek göç sıklık değerlerinin paralel GA'nın performansını bozduğunu göstermiştir. Aynı yıl Cohoon ve arkadaşları, paralel GA'nın işleyişi ile kesintili denge teorisi arasında bazı

benzerlikler kaydetmişlerdir [86]. Tanese devam eden yıllarda göç edecek birey sayısı ve göç sıklık değerlerinin algoritmanın performansına etkisini incelemek için detaylı çalışmalar yapmıştır [87, 88]. Mühlenbein ve arkadaşları tarafından önerilen paralel GA, bazı test fonksiyonlarının küresel en iyi noktasını bulabilen dikkat çekici bir çalışma olmuştur [89]. Bu çalışmada önerilen model bazı araştırmacılar tarafından kendi deneysel çalışmalarında kullanılmıştır [90, 91].

Bir süre sonra, daha fazla araştırmacı uygulama problemlerini çözmek için çok popülasyonlu paralel sezgisel algoritmaları kullanmaya başlamıştır [47, 92–100]. Bu çalışmlarda paralel gerçekleştirim ile daha hızlı ve daha performanslı alternatif modeller oluşturulabilirken, algoritmaların çalışma mantıkları daha iyi anlaşılmaktadır.

Sezgisel algoritmaların karakteristiklerinin birbirlerinden çok farklı olması sebebi ile modelin gerektirdiği parametrelerin değerleri algoritmadan algoritmaya farklılık göstermektedir. Bu amaçla bu parametrelerin farklı algoritmalar için detaylı analiz edildiği çalışmalar da bulunmaktadır [101–104].

Güncel optimizasyon algoritmalarınında sahip oldukları farklı karakteristikler sebebi ile coarse-grained paralel modellerinin performansları ayrıca incelenmektedir. Bu amaçla tez kapsamında incelenen güncel optimizasyon algoritmalarından CS ve FF için çok az sayıda çalışma bulunurken [105, 106], GS ve TLBO algoritmalarının paralel gerçekleştirimleri literatürde henüz bulunmamaktadır.

Ayrıca farklı paralel modellerin bir arada kullanıldığı hybrid çalışmalarda bulunmaktadır [99, 100, 107, 108]. Bu çalışmalarda birden fazla modelin avantajlarının birleştirilebileceği gösterilmektedir.

1.2.5. Uygulamalar

Paralel sezgisel algoritmalar kullanılarak birçok uygulama gerçekleştirilmiştir. Burada sadece temsili bazı örnekler verilmektedir.

Cahon ve arkadaşları paralel kütüphaneler kullanarak evrimsel algoritmalar ve yerel arama algoritmalarının bir arada kullanıldığı yeni bir paralel model önermişlerdir.

Önerilen modelin performansını gerçek dünya problemlerinden radyo ağ tasarım problemi üzerinde incelemişlerdir [109]. Paralel evrimsel algoritmalar ile devre bölümleme problemi [110] ve denetleyici tasarımı [111] gerçekleştirmişlerdir.

Chu ve Zomaya protein katlaması sorunu olarak da bilinen protein yapı tahmini problemi üzerinde iki boyutlu, üç boyutlu ve ayrık hesaplama içeren ACO algoritmasının paralel modelini problemin çözümünde yeni bir yöntem olarak sunmuşlardır [112].

Melab ve arkadaşları klasik temel paralel evrimsel algoritma modellerini işe yararlılık, heterojenlik, geniş ölçeklilik gibi kriterlere göre incelemişlerdir. Ayrıca bu çalışmada paralel evrimsel algoritmalara yönelik genel bir çatı oluşturulabilmesi için metodolojik bir yaklaşım önermişlerdir [113]. Başka bir çalışmada önerdikleri modeli büyük boyutlu kombinatoryal optimizasyon problemleri üzerinde, 100'den fazla işlemci ile test etmişlerdir [114].

Murugavalli ve Rajamani önerdikleri paralel bulanık c-ortalama algoritmasını beyin tümörlerinin bölütlenmesinde kullanmışlardır. Çalışmada tümörlerinin tespit edilmesi için kullanılan resim boyutlarının çok büyük olduğu durumlarda algoritmanın çok hızlı çalışarak zamandan kazanç sağlandığını belirtmişlerdir [115].

Manfrin ve arkadaşları yüksek performanslı bir karınca kolonisi optimizasyonu algoritmasını MPI kütüphanesi kullanılarak paralelleştirmişlerdir. Farklı iletişim topolojilerinin algoritmanın performansına etkisini gezgin satıcı problemi çözümünde incelenmiştir [116].

Talbi ve arkadaşları çok amaçlı optimizasyon problemleri için uygulanabilecek paralel yaklaşımlar hakkında çalışmalar yapmışlardır [117].

Kumar ve Mittal paralel bir bulanık oransal (Fuzzy Proportional, FP), türev (Fuzzy Derivative, FD) ve integral (Fuzzy Integral, FI) denetleyici önermişlerdir. FP + FI + FD denetleyicisinin tepkisini yüksek mertebeden süreçlerde geleneksel paralel PID kontrolör, Ziegler-Nichols ve Åström-Hagglund ayarlama tekniği ile ayarlanmış denetleyiciler ile karşılaştırmışlardır. Deneysel sonuçlarla önerilen paralel FP + FI + FD denetleyicisinin geleneksel PI / PID kontrolörden daha iyi performans gösterdiği

görülmektedir [118].

Bu çalışmaların tamamı geleneksel çok işlemcili paralel mimariler kullanılarak gerçekleştirilmiştir. Çok işlemcili paralel mimarilerin kullanımının nispeten zor olması ve kısmen yüksek maliyetler gerektirmesi nedeniyle araştırmacıların ilgilisi günümüzde farklı donanımlar üzerine kaymaktadır. Özellikle grafik işlemci birimlerinin (Graphic Processing Unit, GPU) genel amaçlı hesaplamalar için kullanımının kolaylaşması, bu donanımlarla yapılan çalışmaları her geçen gün arttırmaktadır. Araştırmacılar bazı esnek hesaplama tekniklerini paralelleştirerek GPU üzerinde gerçekleştirmişler ve muhtelif problemlerin çözümünde başarılı sonuçlar elde etmişlerdir [119–127].

Yapılan bu çalışmalar, zor ve karmaşık problemlerin çözümlerinde esnek hesaplama yöntemleri ile tatmin edici çözümler elde edilebileceğini, bunların paralel modelleri ile de paralel mimarilerin getirdikleri avantajlardan faydalanarak büyük boyutlu zor ve karmaşık problemlerin çözümlerinin gerçekleştirilebileceğini göstermektedir.

Bu tez çalışmasının amaçlarından biri literatürdeki mevcut güncel bazı popülasyon tabanlı sezgisel algoritmaların seri ve paralel modellerini gerçekleştirmek ve bunların parametre analizlerini yaparak başarımlarını karşılaştırmaktır. Bunun yanında gerçekleştirilen algoritmalar için bazı iyileştirme stratejileri önerilerek, performanslarını arttırmak hedeflenmektedir. Bir diğer amaç ise, farklı algoritmaların veya çözüm stratejilerinin dezavantajlarını ortadan kaldırmak amacı ile karma modellerin oluşturulmasıdır. Bu çalışmalarla büyük boyutlu ve karmaşık problemlerin çözümlerini paralel algoritmalar ile gerçekleştirerek, hem zaman hem de performans kazancı sağlamak amaçlanmıştır.

Bu amaçlar doğrultusunda tez şu şekilde bölümlendirilmiştir:

Bölüm-2'de seri ve paralel hesaplama sistemleri, paralel algoritmalar ve paralel programlama hakında bilgiler verilmiştir.

Bölüm-3'te esnek hesaplama yöntemleri hakkında bilgiler verilmiş, yapay sinir ağları, evrimsel hesaplama ve tez kapsamında gerçekleştirilen popülasyon tabanlı sezgisel algoritmalar ayrıntılı bir şekilde anlatılmıştır.

Bölüm-4'de popülasyon tabanlı sezgisel algoritmalarının paralelleştirme modelleri hakkında temel ve teorik bilgiler verilmiştir.

Bölüm-5'de gerçekleştirilen uygulamalar ve elde edilen sonuçlar verilmiştir.

Bölüm-6'da tezden çıkarılan sonuçlar yorumlanmış, tezin katkısı ve ileriye yönelik yapılması planlanan çalışmalardan bahsedilmiştir.

2. BÖLÜM

SERİ ve PARALEL HESAPLAMA

2.1. Giriş

Esnek hesaplama yöntemlerinin paralel programlama teknikleri kullanılarak paralelleştirilmesi ve paralel hesaplama sistemleri üzerinde çalıştırılması ile büyük boyutlu problemlerin kabul edilebilir süreler içerisinde çözümleri gerçekleştirilebilmektedir. Paralel programlama tekniklerinin arkasındaki temel fikir, problemin parçalara bölünmesi ve paralel hesaplama sistemleri ile her bir parçanın çözümünün aynı anda gerçekleştirilmesidir [128]. Paralel hesaplama sistemleri ise, bir problemin parçalarının birkaç işlemci üzerinde aynı anda çözümlenebilmesine olanak sağlayan bilgisayar mimarileridir.

Bilgisayar mimarileri için mantıksal bir sınıflandırma yapmak oldukça zordur. Ancak, Michael J. Flynn tarafından 1966 yılında ortaya konan ve Flynn sınıflandırması olarak bilinen sınıflandırma yaygın olarak kullanılmaktadır [129]. Bu sınıflandırmada hesaplama üniteleri komut ve veri olmak üzere iki bağımsız parametreye göre 4 kategoride incelenmektedir. Şekil 2.1'de Flynn sınıflandırması gösterilmiştir.

Tek komut tek veri akışı (Single Instruction, Single Data stream, SISD) Flynn sınıflandırmasındaki en basit mimaridir. Şekil 2.2(a)'de gösterimi verilen mimaride aynı anda tek bir komut tek bir veriyi işlemektedir [130]. Mimari, bir işlemci ve bir kontrol biriminin olduğu klasik Von Neumann mimarisinin özelliklerini taşır. Klasik Von Neumann mimarisi matematikçi Jon Von Neumann tarafından 1945 yılında bilgisayarların genel yapısını tanımlayan mimaridir ve bellek, kontrol birimi,

	Tek komut	Çok komut
Tek veri akışı	SISD	MISD
Çok veri akışı	SIMD	MIMD

Şekil 2.1. Flynn Sınıflandırması

aritmetik kontrol birimi ve girdi/çıktı birimi olmak üzere dört ana bileşenden oluşmaktadır.

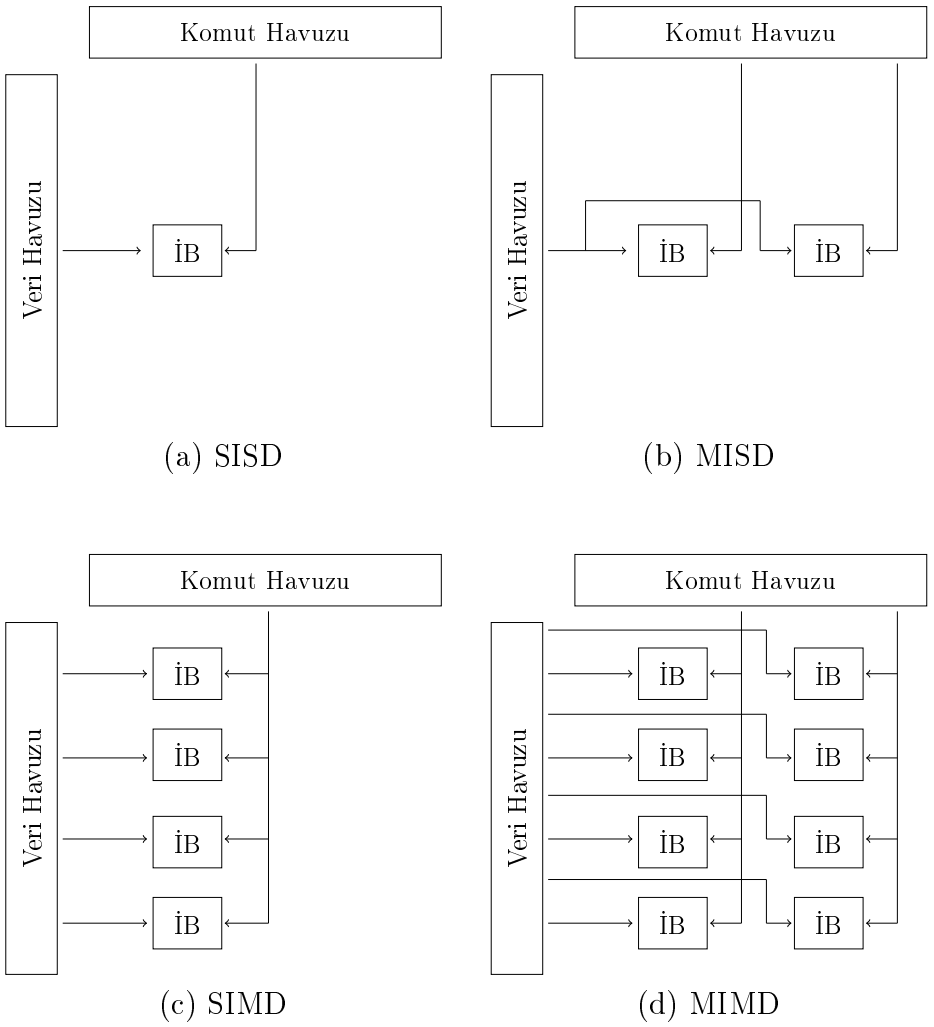
Tek komut çok veri akışı (Single Instruction, Multiple Data streams, SIMD) mimarisinde sistem tek bir kontrol birimine ve birden fazla işlemci birimine sahiptir. Bu sistemde tüm işlemciler aynı anda aynı komutu farklı veriler üzerinde işlerler. Mimaride tüm işlemciler bir kontrol birimi tarafından kontrol edilirler [130]. Şekil 2.2(b)'de SIMD mimarisinin gösterimi verilmiştir.

Çok komut tek veri akışı (Multiple Instruction, Single Data stream, MISD) mimarisinde farklı komut setleri farklı işlemciler ile aynı veri üzerinde işletilmektedir [130]. Bu mimariye sahip olan bilgisayarların tasarlanmasının zor olması ve birçok problem için uygun olmaması sebebi ile yaygın olarak kullanılmamaktadır. Şekil 2.2(c)'de mimarinin gösterimi verilmiştir.

Çok komut çok veri akışı (Multiple Instruction, Multiple Data streams, MIMD) mimarisinde farklı işlemciler farklı komutları farklı veri setleri üzerinde işletebilmektedir [130]. Şekil 2.2(d)'de gösterimi verilen mimaride işlemci ve kontrol birimlerinin sayısı birden fazla olmakta ve işlemciler ara sonuçları birbirlerine iletebilmektedirler. Ayrıca bu mimariye sahip bilgisayarlar belleklere erişim yöntemlerine göre paylaşımlı bellekli MIMD veya dağıtık bellekli MIMD olarak sınıflandırılırlar.

Paralel bilgisayarlar olarak adlandırılan günümüz süper bilgisayarları Flynn'nın

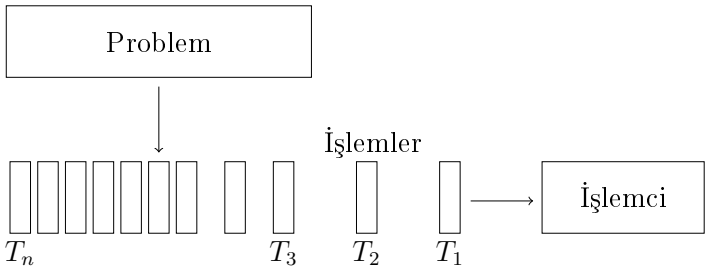
sınıflandırmasına göre SIMD ve MIMD olarak iki temel prensipte tasarlanmaktadır. Bu sistemler çok çekirdekli işlemciler, ortak ağa bağlı birçok bilgisayar, özel bir donanım veya bunların herhangi bir kombinasyonu olabilir.



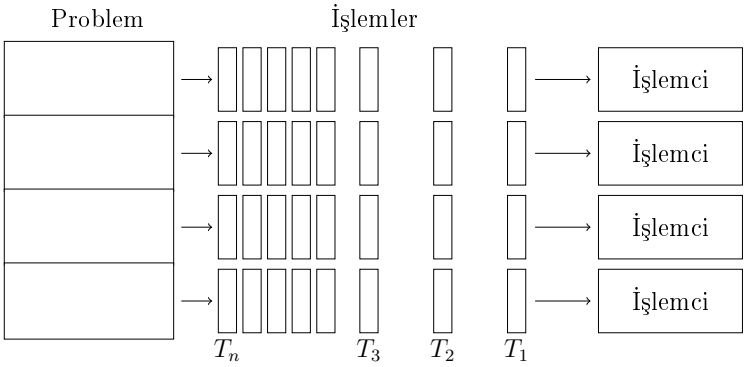
Şekil 2.2. Flynn sınıflandırması işlem birimleri veri havuzu/komut havuzu açısından gösterimi (İB: İşemci birimi)

2.2. Seri Hesaplama

Seri hesaplama, problem çözümlerinin bir bilgisayarda bir işlemci üzerinde gerçekleştirilmesidir. Seri hesaplama ile birim zamanda sadece bir işlem gerçekleştirilir yani bir işlem bitmeden diğer bir işlem çalıştırılamaz ve işlemler sırayla yapılır. Geleneksel yöntemlerde programlar seri hesaplamalar şeklinde yazılır. Seri hesaplama çalışma mantığı, Şekil 2.3'de gösterilmiştir.



Şekil 2.3. Seri hesaplama çalışma mantığı



Şekil 2.4. Paralel hesaplama çalışma mantığı

2.3. Paralel Hesaplama

Paralel hesaplama, problem çözümlerini daha hızlı gerçekleştirebilmek için birden fazla işlemcinin eş zamanlı kullanılmasıdır. Paralel hesaplama yapılabilmesi için problemlerin parçalara bölünebilmesi ve bunların çözümlerinin eş zamanlı olarak koordine edilebilmesi gerekir. Bu sayede büyük sorunların daha kısa sürelerde çözümleri gerçekleştirilebilir. Paralel hesaplama çalışma mantığı, Şekil 2.4'de gösterilmiştir.

Paralel hesaplama, kabaca üç düzeyi içerir. Bunlar paralel hesaplama sistemleri olarak tanımlanan donanımsal düzey, algoritmik düzey ve programlama düzeyidir. Bu düzeyler ile ilgili detaylı bilgiler alt başlıklarda verilmektedir.

2.3.1. Paralel Hesaplama Sistemleri

Paralel hesaplama sistemleri, birden fazla işlemci içeren bir bilgisayarı tanımlamak için kullanılır. Günümüzde kişisel bilgisayarlar üzerinde bulunan çok çekirdekli

işlemcilerden binlerce işlemci içeren sistemlere kadar geniş bir yelpazeyi içine almaktadır. Paralel hesaplama sistemleri genel olarak kullanılan işlemci ve bellek mimarilerine göre sınıflandırılmaktadır.

2.3.1.1. Paralel Hesaplama Sistemleri İşlemci Mimarileri

Paralel hesaplama sistemlerinin işlemci mimarilerini, sıradan paralel bilgisayar mimarileri ve süper bilgisayar mimarileri olarak iki sınıfa ayırmak mümkündür [131].

Çok işlemcili masaüstü bilgisayarlar, simetrik çok işlemcili bilgisayarlar (Symmetric Multiprocessor) ve bilgisayar kümeleri (Cluster of Workstation) sıradan paralel bilgisayar mimarilerine örnek verilebilir. 2, 4 veya 8 işlemcilerden oluşan masaüstü bilgisayarlar, normal masaüstü bilgisayarlara göre daha yüksek performans ile çalışmaktadır. 16, 32 veya 48 işlemciden oluşan simetrik çok işlemcili bilgisayarlar, tek bir ana kart üzerinde birden fazla simetrik işlemcinin bulunduğu paylaşımlı bellek mimarisidir. Çok sayıda bilgisayardan oluşan bilgisayar kümeleri ise dağıtık bellek mimarisidir ve oldukça popülerdir.

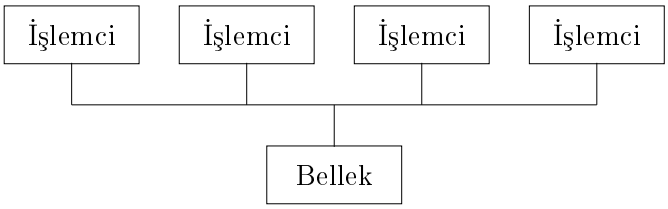
Yoğun paralel işlemciler (Massively Parallel Processor), paralel vektör işlemciler (Parallel Vector Processor) ve dağıtılmış paylaşımlı bellek (Distributed Shared Memory) süper bilgisayar mimarilerine örnek verilebilir. Bu mimariler yüksek bant genişliği ve hızlı iletişim ağı ile birbirine bağlı olan çok sayıda işlemciden oluşan özelleştirilmiş yapılardır.

Sıradan paralel bilgisayar mimarileri ve süper bilgisayar mimarileri karşılaştırıldığında süper bilgisayar mimarilerinin performanslarının çok daha yüksek olduğu söylenebilir. Ancak, maliyetlerinin çok yüksek olması ve üretim sürelerinin çok uzun olması bu mimarilerin birçok alanda kullanımını sınırlamaktadır.

2.3.1.2. Paralel Hesaplama Sistemleri Bellek Mimarileri

Paralel hesaplama sistemlerinin bellek mimarilerini, paylaşımlı bellek, dağıtık bellek ve paylaşımlı-dağıtık bellek olmak üzere üç sınıfa ayırmak mümkündür [128].

Paylaşımlı bellek mimarisinde tüm işlemcilerin küresel adres alanı olarak tanımlanan

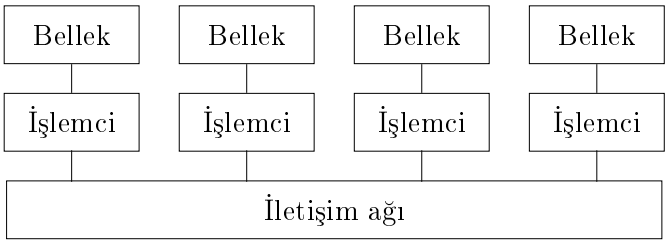


Şekil 2.5. Paylaşımlı bellek

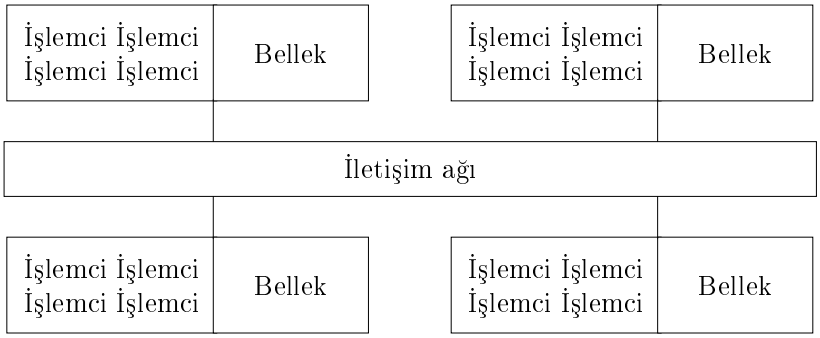
belleklere erişebilme yetenekleri vardır. Bu yapıda işlemciler bağımsız çalışmakta ancak aynı bellek kaynaklarını paylaşmaktadırlar. Dolayısı ile bir işlemci tarafından bellek üzerinde gerçekleştirilen değişiklikler diğer tüm işlemciler tarafından da görülmektedir. Paylaşımlı bellek mimarisi, Şekil 2.5'de gösterilmiştir. Paylaşımlı bellek makineleri belleğe erişim şekillerine göre iki gruba ayrılır [130]. Düzenli bellek erişimi (Uniform Memory Access, UMA) eşit zamanlı ve eşit miktarda belleğe erişim yapan işlemcilerden oluşur. Simetrik çok işlemcili bilgisayarların bellek erişimleri örnek verilebilir. Düzensiz bellek erişimi (Non-Uniform Memory Access, NUMA) bellek erişim zamanının belleğin işlemci üzerindeki yerine bağlı olduğu bir bilgisayar belleği tasarımıdır. Fiziksel olarak birbirine bağlı iki veya daha fazla sayıda simetrik çok işlemcili bilgisayarlar örnek verilebilir. Bu yapıda işlemciler eşit erişim zamanları ile tüm belleklere erişemez, kendi yerel belleğine yerel olmayan bellekten daha hızlı erişir.

Dağıtık bellek mimarisinde tüm işlemcilerin kendi yerel bellekleri vardır. İşlemciler kendi bellekleri dışında diğer işlemcilerin belleklerine erişip kullanamazlar, yani işlemciler arasında küresel adres alanı kavramı yoktur. Her işlemcinin kendi yerek belleği olduğu için birbirlerinden bağımsız çalışırlar, bellekler üzerinde yapılan değişiklikler diğer işlemciler tarafından görünmez. Bir işlemci başka bir işlemcinin verilerine erişim ihtiyacı duyduğunda, nasıl ve ne zaman iletişim kurulacağı genellikle programcılar tarafından ayarlanır [128]. Görevler arası senkronizasyon da yine programcıların sorumluluğundadır. Paylaşımlı bellek mimarisi Şekil 2.6'da gösterilmiştir.

Paylaşımlı bellek mimarisi ve dağıtık bellek mimarisi karşılaştırıldığında, paylaşımlı bellek mimarisinde programlamanın kolay ve veri paylaşımının hızlı olduğu söylenebilir. Dağıtık bellek mimarisinde ise işlemciler arası iletişim, veri transferleri



Şekil 2.6. Dağıtık bellek



Şekil 2.7. Paylaşımlı-Dağıtık bellek

gibi birçok detaydan programcı sorumlu olduğu için programlama daha zordur, aynı zamanda uzak bir düğüm üzerinde bulunan verilere erişmek daha uzun sürer. Buna karşın, dağıtık bellek mimarisinde işlemci ve bellek arasındaki ölçeklenebilirlik paylaşımlı bellek mimarisinde mümkün değildir. Yani işlemci sayısı ve bellek orantılı arttırmak dağıtık bellek mimarisinde mümkünken paylaşımlı bellek mimarisinde maliyetli ve sınırlıdır.

Her iki bellek mimarisinin bir arada kullanıldığı yapılara paylaşımlı-dağıtık bellek mimarisi denilmektedir. Paylaşımlı bellek makineleri paylaşımlı bellek bileşeni olabilirken, birden çok paylaşılan bellek dağıtık bellek bileşeni olabilir. Günümüzde çok hızlı bilgisayarlar bu mimari ile gerçekleştirilmektedir. Bu sayede hem paylaşımlı bellek hem de dağıtık bellek mimarilerinin avantajlarından faydalanılmaktadır. Paylaşımlı-dağıtık bellek mimarisi Şekil 2.7'de gösterilmiştir.

2.3.2. Paralel Algoritmalar

Algoritma, verilen bir problemi çözmek için uygulanan bir yöntem ve izlenecek adımlardır. Algoritmalar, gerçekleştirimleri, çalışma alanları, karmaşıklıkları gibi

farklı özelliklerine göre sınıflandırılabilirler. Seri ve paralel algoritmalar şeklinde yapılan sınıflandırma algoritmaların gerçekleştirimlerine göre yapılmaktadır.

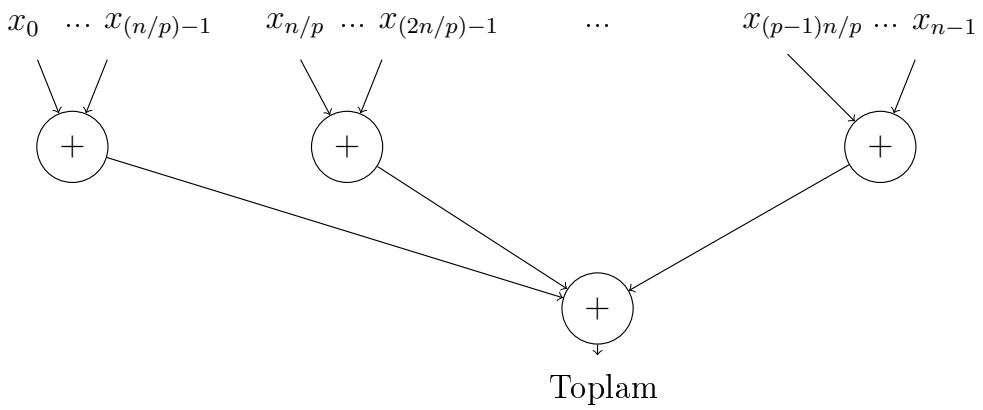
Seri algoritmalar seri hesaplama sistemleri üzerinde, paralel algoritmalar paralel hesaplama sistemleri üzerinde, belirli bir problemin nasıl çözümleneceğini gösteren işlem adımlarıdır. Ancak paralel bir algoritma geliştirmek sadece işlem adımlarını belirlemekten çok daha karmaşıktır. Paralel algoritmalar için ilk olarak çözümün paralelleştirmeye uygun olup olmadığına bakılmalı, daha sonra eş zamanlı çalışacak işlem adımları belirlenmeli ve işlemciler arası senkronizasyon planlanmalıdır. Ayrıca, mümkün olduğunca işlemciler arası haberleşmelerden kaçınılarak, en uygun işlemci sayısı belirlenmelidir. Genellikle tasarlanan paralel algoritmalardan aşağıdaki belirtilen kriterlerden tamamını veya bir kısmını karşılaması beklenir [128].

- Eş zamanlı yapılabilir iş bölümlerinin belirlenmesi
- İş parçacıklarının paralel çalışacak süreçlere dönüştürülmesi
- Programla ilgili girdi, çıktı ve ara verilerin dağıtımı
- Birden fazla işlemci tarafından paylaşılan verilere erişimin yönetilmesi
- İşlemciler arası senkronizasyon

Bu kriterlerin başarılı bir şekilde gerçekleştirilebilmesi için öncelikle verilen problemin çok iyi bir biçimde anlaşılması gerekir. Ayrıca bu kriterlerin her birinin gerçekleştirimi için birkaç seçenek bulunmaktadır. Dolayısı ile farklı seçenekler, farklı paralel mimariler üzerinde veya farklı paralel programlama dilleri ile farklı performans gösterebilmektedirler.

2.3.2.1. Paralel Algoritmaların Tasarım ve Uygulanması

Paralel algoritma tasarlanırken aynı anda işlenecek komutlar belirlemeli ve bu komutlar koordine edilmelidir. Bu tasarımlar için algoritmaların seri modelinin incelenerek paralelleştirilebilir kısımlarının tespit edilmesi yada seri modeline bakılmaksızın yeni bir paralel algoritma geliştirilmesi tercih edilebilir. Her iki



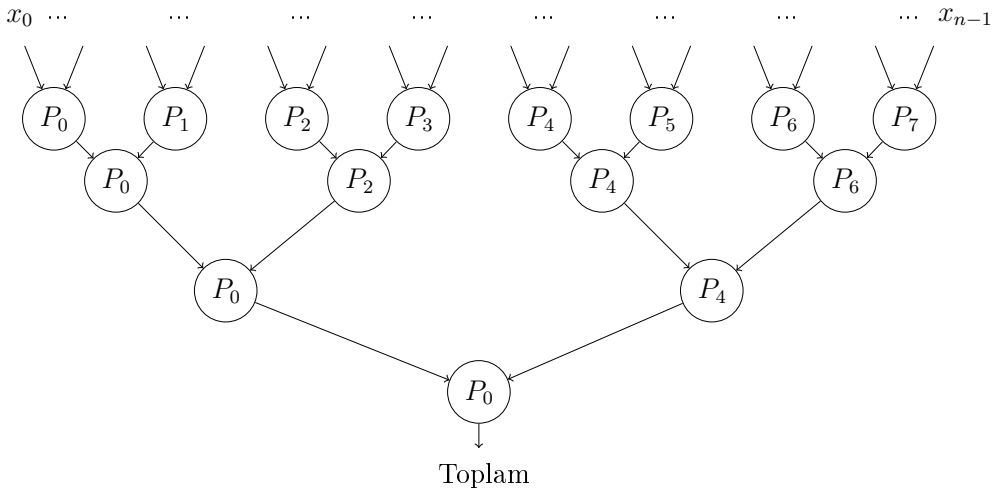
Şekil 2.8. Bölümleme metodu

durumda da algoritmanın doğru tasarlanması paralel hesaplamadan sağlanacak kazancı arttıracaktır. Paralel algoritmaların tasarımı için farklı metotlardan bahsedilebilir. Bu metotlardan bölümleme (Partitioning), böl ve yönet (Divide and conquer) ve iş hattı (Pipelining) yaygın olarak kullanılmaktadır [132].

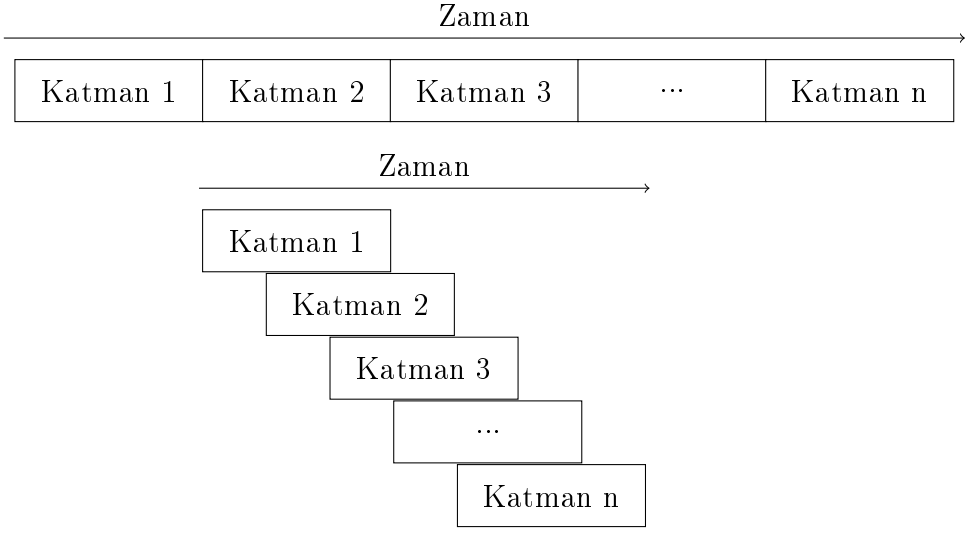
Bölümleme metodu, verilen bir problemi hemen hemen eşit büyüklükte özdeş olmayan birkaç alt probleme parçalayan ve bu alt problemleri eş zamanlı olarak çözen bir metottur [132]. Bölümleme metodu Şekil 2.8'da gösterilmiştir. Böl ve yönet metodu, orijinal problemi aynı formda alt problemlere bölen, bölünen alt problemleri özyinelemeli olarak çözen ve bu çözümlerin birleştirilmesi ile orijinal problemin çözümünü elde eden bir metottur [132]. Böl ve yönet metodu Şekil 2.9'da gösterilmiştir. İş hattı metodu, büyük bir problemin küçük alt problemlere bölünmesi ve bu alt problemlerin kendi aralarında paralel olarak yürütülmeleridir. Çok tekrarlanan ve alt parçalara bölünebilen problemlerde kullanılır. Örnek olarak bir fabrikadaki üretim montaj hattı düşünülebilir. İş hattı metodu Şekil 2.10'de gösterilmiştir.

Tasarlanan paralel algoritmalar paralel hesaplama sistemleri üzerinde uygulanmak istendiğinde görevlerin ayrıştırılması, atanması iletişimlerinin planlanması ve senkronizasyon gibi bazı problemler ortaya çıkmaktadır.

Ayrıştırma yapılırken bilgisayarların görevlerinin dengeli dağılmasına ve ilgili görevler arasındaki iletişimin az olmasına dikkat edilmelidir. Paralel süreçlerde



Şekil 2.9. Böl ve yönet metodu



Şekil 2.10. İş hattı metodu

problemin ayrıştırılmasının iki yolu bulunmaktadır. Bunlar veri kümesi ayrıştırması (Domain Decomposition) ve fonksiyonel ayrıştırmadır (Functional Decomposition) [130]. Veri kümesi ayrıştırmasında veri parçalara bölünmekte, her bir paralel süreç verinin belli bir parçası üzerinde çalışmaktadır. Fonksiyonel ayrıştırma ise yapılacak işler üzerinden olmaktadır. Bu ayrıştırma da problem çok sayıda alt fonksiyona bölünmekte ve her bir paralel sürecin bu alt fonksiyonlardan birini yapması sağlanmaktadır. Ayrıştırma adımından sonra alt problemlerin uygun işlemcilere atanması ve yürütme sırasına karar verilmesi gerekir. Yük dengeleme olarak da bilinen bu adımda yapılacak işler tüm işlemciler her zaman meşgul olacak şekilde dağıtılmaya çalışılır. Algoritma performansında önemli bir rol oynayan bu adım aslında bir çizelgeleme sorunudur. Bu sorun eşit kapasiteye sahip işlemciler ile çalışırken kolaylıkla aşılabilirken, farklı kapasiteye sahip işlemcilerle çalışırken bazı analiz araçları ile ölçümler yapılması ve farklı çizelgeleme tekniklerinin kullanılması gerekebilir.

Paralel çalışan görevler arasında iletişimin gerekliliği, şekli ve sıklığı tasarlanan algoritmaya bağlıdır. Bazı uygulamalar işlemciler arası hiç bir iletişim olmadan gerçekleştirilebilirken birçok uygulama görevler arası veri paylaşımına ve iletişime ihtiyaç duyar. İşlemciler arası iletişim uygulamaya ek bir yük getirdiğinden algoritma tasarımı esnasında mümkün olduğu kadar azaltılmaya çalışılır. İletişim iki işlemci arasında olabileceği gibi bir işlemci ile bir grup işlemci arasında da olabilir.

Görevler arası senkronizasyon problemini çözmek için kullanılan farklı mekanizmalar bulunmaktadır. Bu teknikler genel olarak görevlerin senkronize olmasını sağlarken aynı kaynağa aynı anda erişim gibi kilitlenme durumlarını da engellerler. En yaygın tercih edilen mekanizmalar bariyer, kilit/semafor ve eş zamanlı erişimdir [133]. Bariyer mekanizmasında paralel görevler bariyer adı verilen senkronizasyon noktalarında birbirlerini beklerler. Tüm görevler bariyer noktasına ulaşınca devam edilir. Kilit/semafor mekanizmasında birden fazla paralel görev tarafından erişilen kaynaklar kontrol edilir. Eğer bir görev bir kaynağa erişmiş ise kaynak kilitlenir ve diğer görevlerin erişimi engellenir. Erişen görev işini bitirince kaynak tekrar kilitsiz duruma alınır. Eş zamanlı iletişimde ise birbirleri ile haberleşen görevler arasında verilerin alınıp, gönderilmesi süresince her iki tarafın işlemlerinin bitene kadar bloke

olması durumudur [133].

2.3.2.2. Paralel Algoritmaların Performanslarının Değerlendirilmesi

Paralel algoritmaların performanslarını ölçmek için kullanılan çeşitli yöntemler ve ölçümler bulunmaktadır. Hızlanma, verimlilik ve ölçeklenebilirlik bu yöntem ve ölçümlerin en önemlileridir [128].

Hızlanma, problem çözümünün tek bir işlemci ile gerçekleştirildiğinde harcanan zamanın paralel hesaplama sistemleri ile gerçekleştirildiğinde harcanan zamana oranı olarak tanımlanır ve Eşitlik (2.1)'deki gibi hesaplanır [128]. Verimlilik ise ne kadar işlemci ile ne kadar hızlanma sağlandığının ölçüsüdür ve Eşitlik (2.2)'deki gibi hesaplanır [128]. İdealde elde edilen hızlanma değerinin kullanılan işlemci sayısına eşit, verimlilik değerinin ise 1'e eşit olması beklenir. Ancak pek çok durumda bu mümkün olamamaktadır. İşlemci sayısının artması ile işlemciler arası iletişim ihtiyacının artması, iletişim yollarında yaşanacak gecikmeler ve programın paralelleştirilemeyen kısımları hızlanma ve verimliliği etkileyen en önemli faktörlerdir.

$$\text{Hızlanma} = \frac{\text{Zaman}_{\text{1 işlemci}}}{\text{Zaman}_{\text{m işlemci}}} \tag{2.1}$$

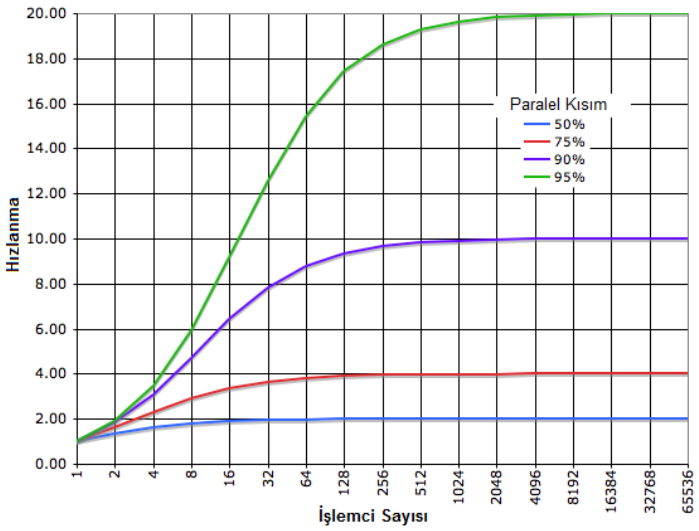
$$\text{Verimlilik} = \frac{\text{Hızlanma}}{\text{m}} \tag{2.2}$$

Bir örnek üzerinden açıklamak gerekirse; %80 paralelleştirilebilen bir program seri olarak 1 işlemci üzerinde 10 birim zamanda çalıştığını düşünelim. Bu program 8 işlemci üzerinde çalıştırıldığında programın paralel kısmı 1 birim zaman, seri kısmı 2 birim zaman olmak üzere toplam 3 birim zaman harcanacaktır. Bu program 16 işlemci üzerinde çalıştırıldığında paralel kısmı 0.5 birim zaman, seri kısmı yine 2 birim zaman olmak üzere toplam 2.5 birim zaman harcanacaktır. Görüldüğü gibi işlemci sayısını 8 'e çıkardığımızda 3.33 kat hızlanma elde ederken işlemci sayısını 16'ya çıkardığımızda sadece 4 kat hızlanma elde edilebilmiştir. Ayrıca bu hesaplama yapılırken işlemciler arası iletişim maliyeti ve diğer kısıtlamalar dikkate alınmamıştır.

Verilen bu hesaplama örneği Amdahl'ın ünlü yasasına dayanmaktadır. Bu yasaya göre bir programın paralelleştirilmesi ile elde edilecek hızlanma o programın seri kısımlarının programda ne kadar yer kapladığına bağlıdır [134]. Amdahl yasasının matematiksel gösterimi Eşitlik (2.3)'de verilmiştir.

$$\psi \leq \frac{1}{f + \frac{1-f}{p}} \tag{2.3}$$

Burada, ψ hızlanma, p işlemci sayısı, f programın paralelleştirilemeyen kısmının programın toplam çalışma zamanına oranıdır ve $0 \leq f \leq 1$ arasında bir değerdir. Amdalh yasasına göre paralelleştirme sadece az sayıda işlemci için ya da f'in çok küçük olduğu problemlerde kullanışlıdır. Bu yüzden paralel programlama ile f değeri mümkün olduğu kadar küçük tutulmaya çalışılır. Şekil (2.11)'de farklı f değerleri için elde edilebilecek hızlanma değerlerinin grafiksel gösterimi verilmiştir [135].



Şekil 2.11. Amdahl yasası

Paralel hesaplama bilimindeki bir diğer yasa Gustafson yasasıdır [136]. Amdahl yasası ile yakından ilişkili olan bu yasa, yeterince büyük bir problemin verimli bir biçimde paralelleştirilebileceğini öngörmektedir. Gustafson yasasının matematiksel gösterimi Eşitlik (2.4)'de verilmiştir.

$$\psi \leq p + (1 - p)s \tag{2.4}$$

Burada, ψ hızlanma, p işlemci sayısı, s programın paralelleştirilemeyen kısmıdır. Gustafson yasası, büyük ölçekli paralel hesaplama sistemlerinin ölçümlerini karşılamayan Amdahl yasasının yeniden değerlendirilmesidir. Bu yasa paralel işlemciler üzerindeki sabit hesaplama yükünü kaldırarak yerine ölçekli hızlanmayı sağlayan sabit zaman kavramını getirmektedir.

Amdahl ve Gustafson yasaları karşılaştırıldığında özetle Gustafson yasası, programların paralelleştirilmeyen kısmının çok sayıda işlemcinin kullanıldığı paralel sistemlerde bile sabit kaldığı, Amdahl yasası ise programların paralelleştirilemeyen kısmının başarım üzerindeki etkisinin işlem sayısıyla doğru orantılı biçimde artış gösterdiğidir.

Ölçeklenebilirlik, bir bilgisayar sisteminin performans ve işlevsellik talebini karşılamak için artan kaynak üzerinde göstermiş olduğu sonuçlarla ilgilidir. Ölçeklenebilirliğin farklı boyutları bulunmaktadır. Örneğin makine boyutu ölçeklenebilirlik, ilave işlemciler ile performansın ne kadar iyileştirildiği ile ilgilenir ve sistemler için kullanılabilecek maksimum işlemci sayısını belirlenmesinde kullanılır. Problem boyutu ölçeklenebilirlik, artan veri ve problem boyutu için kullanılan sistemlerin problemin üstesinden gelip gelemeyeceği ile ilgilenir. Teknolojik ölçeklenebilirlik değişen teknoloji ile ne kadar iyi performans artışı elde edildiği ile ilgilenir. Verimlilik, hızlanma ve ortalama gecikme ölçeklenebilirlik çalışması için kullanılan ölçümlerdir.

2.3.3. Paralel Programlama

Paralel programlama, performansı arttırmaya yönelik yapılan işlemlerin paralel hesaplama sistemleri üzerinde çalışmasını sağlayan yapılardır. Paralel hesaplama sistemlerinin gelişimi farklı paralel programlama modellerinin, dillerinin ve metotlarının gelişiminde beraberinde getirmiştir. Bu amaçla işletim sistemi seviyesinde ve programlama dili seviyesinde pek çok yazılım sistemi geliştirilmiştir.

2.3.3.1. Paralel Programlama Modelleri

Paralel programlama modelleri paralel program yazmak için kullanılan modellerdir. Bu modeller, uygulamalar, diller, derleyiciler, kütüphaneler ve iletişim sistemleri olabilir. Araştırmacılar uygulamaları için uygun bir model veya karma bir model seçerek, uygulamalarını gerçekleştirebilirler. Uygun model seçilirken genellikle modelin genellenebilirliği dikkate alınır. Model ne kadar farklı problemleri ifade edebilir ve farklı mimariler üzerinde gerçekleştirilebilirse o kadar ön plana çıkmaktadır.

Literatürde yaygın olarak kullanılan paralel programlama modelleri mesaj aktarma ve değişken paylaşımlı modellerdir [130].

Mesaj aktarma modelinde her bir işlemcinin bellek için kendi adres alanı vardır. Bir işlemci bir başka işlemcinin verilerine doğrudan ulaşamayabilir. Bu nedenle işlemciler arası haberleşme mesajların gönderilip alınması ile sağlanır [132]. Veri transferleri esnasında her iki işlemci ortak çalışır. Genelde büyük çaptaki bilgisayar sistemlerinde kullanılmaktadır. Bu modelde en iyi bilinen ve kullanılan kütüphane Message-Passing Interface (MPI) kütüphanesidir [137, 138].

Değişken paylaşımlı modelde tüm işlemcilerin belleğe erişimini destekleyen ortak bir adresleme alanı vardır. Bu tür sistemlerde veriler bu ortak alanda paylaşılır, eş zamanlı olmadan okuma ve yazma işlemleri bu ortak alan üzerinde yapılabilir [128]. Genelde simetrik çok işlemcili bilgisayar sistemlerinde kullanılmaktadır. Verilerin ortak bir alanda paylaşılması sebebi ile yönetilmesini kısmen zorlaştırmaktadır. Bu modelde en iyi bilinen ve kullanılan kütüphane POSIX (PThreads) kütüphanesidir [139].

2.3.3.2. Paralel Programlama Dilleri ve Paralelleştirme Metotları

Paralel programların tasarım ve gerçekleştirimi için farklı yollar izlenebilir. Bunlar, programları otomatik paralelleştiren derleyicilerin kullanılması, yeni bir paralel programlama dili tercih edilmesi yada mevcut programlama dillerine paralel programlama komutlarını algılayan MPI, OpenMP gibi kütüphanelerin eklenmesi

şeklinde olabilir.

Otomatik paralelleştiren derleyiciler, programın kaynak kodunu analiz ederek paralelleştirilebilir kısımları belirleyerek yada programcıların direktifleri ile paralelleştirme yaparlar [130]. Bu derleyiciler genellikle koddaki döngüler üzerine yoğunlaşırlar. Ancak yapılan çalışmalar, otomatik paralelleştirme yapan derleyicilerin performansının yeteri kadar etkili olmadığı yönündedir. Yeni geliştirilen paralel programlama dilleri ise güncel paralel makinelerin hesaplama güçlerinden daha fazla yararlanılmasına odaklanmışlardır. Bu sebeple kullanım kolaylığı ve kod geliştirme verimliliği ikinci plana atılmıştır. Co-array Fortran ve UPC bu amaçla geliştirilmiş paralel programlama dillerine örnek verilebilir.

Seri programlama dillerine paralel kütüphanelerin eklenmesi, paralel programlama gerçekleştirmek için kolay ve popüler bir yoldur. MPI ve OpenMP kütüphaneleri bu konuda en yaygın kullanılan kütüphanelerdir.

3. BÖLÜM

ESNEK HESAPLAMA YÖNTEMLERİ

3.1. Giriş

Esnek hesaplama, polinomsal zamanda tam çözüm üretilemeyen, hesaplaması zor, belirsizlik içerebilen, tam olarak ifade edilemeyen, analitik çözümü bulunmayan, problemlerin çözümleri için geliştirilmiş belirsizlik, kısmi doğruluk, yakınsaklık toleransı bulunan tekniklere genel olarak verilen isimdir [140].

Klasik ve geleneksel yöntemlerle çözümü elde edilemeyen problemlere esnek hesaplama yöntemleri ile çözüm üretilebilmekte, yorum getirilebilmektedir. Bulanık mantık, yapay sinir ağları ve evrimsel hesaplama gibi teknikler esnek hesaplama yöntemlerine örnek verilebilir.

Problem tanımlarında 1 ve 0'lara dayalı klasik mantığın yetersiz kalması, bulanık mantık kavramının ortaya çıkmasına neden olmuştur [141]. Bulanık mantık kavramını, belirsizliklerin anlatımı ve belirsizliklerle çalışılabilmesini sağlayan matematiksel düzen olarak tanımlamak mümkündür.

Yapay sinir ağları insan beyninin çalışma mantığından esinlenilerek geliştirilmiştir. Bu teknik ile insanların sahip olduğu düşünebilme ve öğrenebilme yetenekleri benzetilmeye çalışılır [142].

Evrimsel hesaplama biyolojik evrim ilkelerine dayalı problem çözme tekniklerine verilen ortak isimdir [143]. Çoğunlukla karmaşık problemlerin çözümleri için geliştirilmiş bu teknikler, optimizasyon algoritmalarını içerir. Optimizasyon algoritmaları herhangi bir amacı gerçekleştirmek için çeşitli alternatif hareketlerden

etkili olanları seçme yöntemleridir. Birçoğu rassaldır ve problemlerin kesin çözümünü elde edeceklerini garanti edemez, ama sahip oldukları yakınsama özellikleri sayesinde makul bir süre içerisinde kesin çözüme yakın bir çözümü garanti edebilirler [144, 145].

Tez kapsamında gerçekleştirilen yapay sinir ağları ve popülasyon tabanlı sezgisel algoritmalar aşağıda ayrıntılı olarak anlatılmaktadır.

3.2. Yapay Sinir Ağları

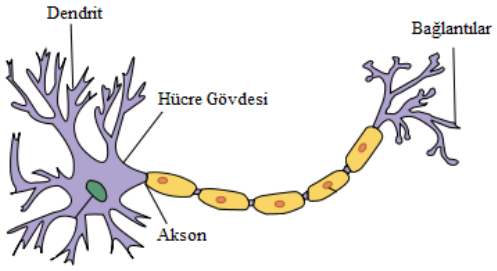
Yapay Sinir Ağları (YSA) insan beyninin fonksiyonel özelliklerini modellemeyi amaçlayan programlardır. Ancak insan beyninin çalışma mantığı oldukça karmaşıktır ve tam anlamıyla modellenmesi imkansızdır. YSA ile sadece öğrenebilme kabiliyeti gibi temel birkaç özelliği modellenmeye çalışılır [142]. Bu özellikler sayesinde öğrenebilen, ilişkilendirebilen, sınıflandırabilen ve genelleme yapabilen uygulamalar gerçekleştirilebilmektedir.

3.2.1. Sinir Hücresi Modeli

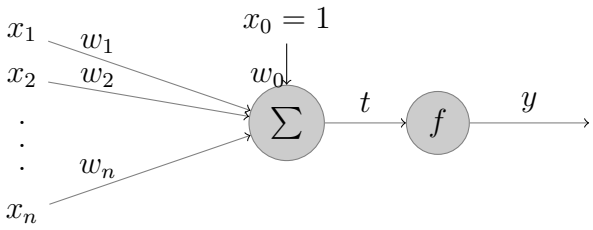
Birçok YSA modelinin tasarlanmasında biyolojik sinir sistemleri temel alındığı için öncelikle biyolojik sinir sistemini oluşturan sinir hücresi tanıtılacaktır.

Sinir sisteminin merkezini oluşturan ve ortalama 1.5 kilogram ağırlığındaki insan beyni çok sayıda sinir hücresinden (nöron) oluşmaktadır. Öğrenme, hatırlama, düşünme ve algılama gibi tüm bilişsel davranışları da içeren her türlü insan davranışının temeli bu nöronlardır. Nöronlar birbirlerine karmaşık bir şekilde bağlantılıdır ve bu bağlantılı yapıya ağ adı verilir.

Sinir hücrelerinin doğuştan sahip olunması ve insanın yaşamı boyunca yenilenmemesi en belirgin özellikleridir. Beynin gelişmesi ve ağırlık kazanması sinir hücrelerinin büyümesi ve aralarında yeni bağlantıların kurulmasından kaynaklanmaktadır. Sinir sistemi içerisindeki farklı işlevlere sahip sinir hücreleri olmakla birlikte her sinir hücresi Şekil 3.1'deki gibi hücre gövdesi (soma), dentrit (dendrite), akson (axon) ve bağlantılardan (synapse) meydana gelir [146].



Şekil 3.1. Biyolojik sinir hücresi



Şekil 3.2. Yapay sinir hücresi

Dentritler, hücre gövdesinin saça benzeyen uzantılarıdır ve hücreye gelen sinyalleri toplarlar. Hücre gövdesi, gelen bu sinyalleri işleyerek bir çıkış sinyali üretilip üretilemeyeceğine karar verir. Eğer çıkış sinyali üretilecekse bu çıktıyı akson ve bağlantılar aracılığıyla diğer nöronlara veya organlara gönderir [142].

3.2.2. Yapay Sinir Ağı Hücresi

Yapay sinir ağı hücre modeli de gerçek biyolojik hücreyle aynı ilkelere dayandırılarak doğal nöronların temel fonksiyonlarını simule etmeye çalışır. Ancak yapay sinir ağı hücresi biyolojik sinir sistemlerinden çok daha basittir. Şekil 3.2'de yapay nöronun genel yapısı verilmiştir. Bu yapıda YSA'ların üç ana elemanı görülmektedir. Bunlar temel işlem elemanı nöron, giriş ve çıkış verileri arasındaki bağlantı ve bu bağlantıların sağlamlığını gösteren bağlantı ağırlıklarıdır [142]. Ayrıca sisteme eklenebilen eşik değeri de nörona giriş olarak gösterilmiştir.

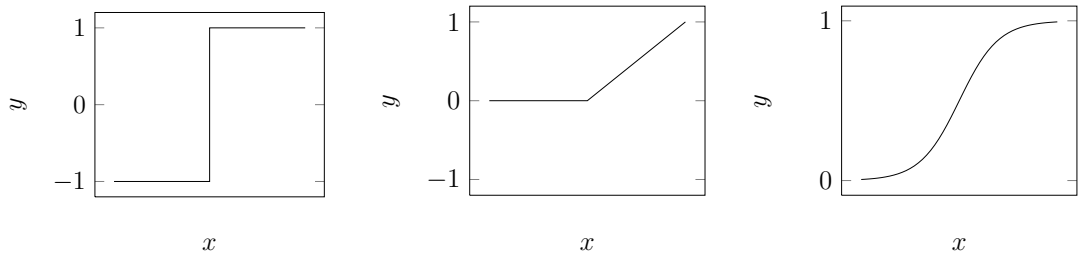
Burada $i = 0, 1, 2, ..., n-1$ olmak üzere, giriş sinyalleri x_i ve her bir giriş sinyalinin bağlantı ağırlığı w_i sembolleri ile gösterilmiştir. Hücre gövdesi ise tüm girdi sinyallerinin ağırlıklı toplamlarını ve eşik değerini almaktadır. Tüm bu toplam sinyal

t ile gösterilmiş ve sinapsise transfer fonksiyonuna girdi olarak yönlendirilmiştir. Sinapsis üzerindeki transfer fonksiyonundan çıkan sonuç sinyali y ile belirtilmiş ve diğer hücreye beslenmek üzere yönlendirilmiştir. Oluşturulan bu modelin matematiksel gösterimi Eşitlik (3.1)'deki gibidir.

$$y = f(g(x)) = f((\sum_{i=0}^{n} w_i x_i)) \tag{3.1}$$

Bu gösterimde, y çıkış değerini, $f()$ transfer fonksiyonunu, g() toplama fonksiyonunu, x_i giriş değerlerini ve w_i bağlantı ağırlıklarını göstermektedir.

YSA'larda çok çeşitli transfer fonksiyonları kullanılabilmektedir [142, 147]. Şekil 3.3'de yaygın kullanılan bazı transfer fonksiyonları verilmiştir. Sigmoid transfer fonksiyonu (Şekil 3.3(a)) en fazla tercih edilen transfer fonksiyonudur. Eşitlik (3.2)'de denklemi verilen sigmoid transfer fonksiyonu, g() toplama fonksiyonundan gelen değeri alıp sıfır ile bir arasında bir değere dönüştürür. Bu değer yapay nöronun çıktısıdır, YSA'nın çıktısı olarak dış ortama yada bir başka nörona giriş olarak iletilebilir.

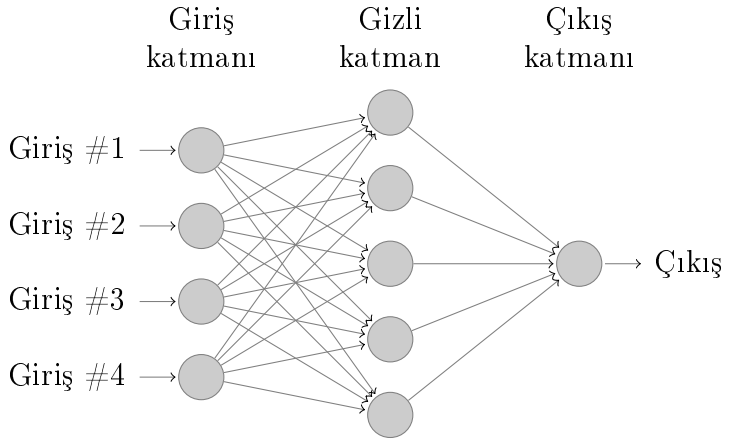


(a) Step (b) Threshold (c) Sigmoid

Şekil 3.3. Yaygın kullanılan bazı transfer fonksiyonlar

$$f(x) = \frac{1}{1 + \mathrm{e}^{-x}} \tag{3.2}$$

Yapay nöronun görevi x_i giriş sinyallerine karşılık y çıkış sinyalinin oluşturulmasıdır. Bu amaçla w_i ağırlıkları ve b eşik değeri her bir giriş ve çıkış sinyalleri arasındaki ilişkiyi göre tekrar tekrar ayarlanır. Bu süreç öğrenme süreci olarak adlandırılır. Öğrenme süreci tanımlanan bir optimum noktaya gelindiğinde sonlandırılır.



Şekil 3.4. Genel bir MLP ağı

YSA'lar yapay sinir hücrelerinin birleşiminden oluşan katmanlı yapının tümü olarak nitelendirilir. Günümüzde, farklı amaçlar ve değişik uygulamalar için birçok yapay sinir ağı modeli geliştirilmiştir [148, 149]. Bu modeller arasında en fazla tercih edilen çok katmanlı ileri beslemeli yapay sinir ağıdır (Multiple Layer Perceptron, MLP). Şekil 3.4'de genel bir MLP ağının yapısı gösterilmektedir. Bu yapıda bazı nöronlar giriş sinyallerini alır (giriş katman), bazı nöronlar çıkış sinyallerini iletir (çıkış katman), diğerleri de ara katmanda (gizli katman) bulunur ve sadece ağ içerisinde bağlantıları vardır. Giriş ve çıkış katmanları tek tabakadan oluşurken, bu iki katman arasında birden fazla ara katman bulunabilir.

YSA'ların performansları hesaplanan hata değeri ile ölçülür. Hata değeri hesaplanırken, ağın çıkış değeri, istenen çıkış değeri ile karşılaştırılır. Bu amaçla kullanılan farklı hata hesaplama formülleri bulunmaktadır. Ağın çıkış değeri ve istenen çıkış değerinin farkının karesinin alındığı ortalama karesel hata formülü yaygın olarak kullanılmaktadır. Öğrenme süresi boyunca, güncellenen bağlantı ağırlık değerleri ve eşik değerleri ile hesaplanan hata değeri düşürülmeye çalışılır.

3.2.3. Yapay Sinir Ağları Çeşitleri

YSA'lar öğrenme yöntemleri, ağ yapıları, kullandıkları veri setleri, öğrenme zamanları gibi özelliklerine göre farklı şekillerde sınıflandırılabilirler [142, 147, 150].

YSA, öğrenme yöntemlerine göre; denetimli, denetimsiz ve destekleyici öğrenme olmak üzere üçe ayrılır. Denetimli öğrenmede YSA'lar hem giriş değerleri hem de

girişlere karşılık gelen çıkış değerlerinden oluşan bir veri seti ile eğitilir. Ağ bu veri seti üzerinden verilen girişe karşılık istenen çıkış değerinin elde edilebilmesi için bağlantı ağırlıklarını ve varsa eşik değerlerini en uygun değerlere ayarlamaya çalışır. Daha sonra bu değerler ile ağa verilen benzer girişler için çıkış değerleri üretir. Denetimsiz öğrenmede ise sistemin öğrenmesine yardımcı olan herhangi bir çıkış bilgisi yoktur. Verilen örnek giriş değerleri arasındaki ilişkilerden kendi sınıflandırmasını yaparak kendi kurallarını oluşturur. Destekleyici öğrenmede ise giriş değerlerine karşılık gelen istenen çıkış değerleri verilmez, sistem kendi çıktısını ürettikten sonra bu çıktının iyi veya kötü olup olmadığına dair bir bilgi verir.

YSA, ağ yapılarına göre; ileri ve geri beslemeli olmak üzere ikiye ayrılır. İleri beslemeli ağlarda nöronlar girişten çıkışa doğru düzenli katmanlar şeklindedir. Bir katman sadece kendisinden sonraki katmanla bağlantılıdır. Yani YSA gelen bilgileri sırası ile giriş katmanı, gizli katman ve çıkış katmanında işler. Geri beslemeli ağlarda ise bir hücrenin çıktısı kendinden önceki, yada kendi katmanındaki herhangi bir hücreye girdi olarak verilebilir.

YSA, öğrenme zamanına göre; statik ve dinamik öğrenme olmak üzere ikiye ayrılır. Önce eğitilip, sonra kullanılan YSA'lar statik öğrenme kuralı ile çalışırlar. Eğitim işleminin çalışılan süre boyunca devam ettiği YSA'lar ise dinamik öğrenme kuralı ile çalışır.

3.3. Popülasyon Tabanlı Sezgisel Algoritmalar

Algoritmaların geliştirilmesinde genellikle sosyal, biyoloji, fizik, bilgisayar gibi bilimler temel alınmaktadır. Geliştirilen bu algoritmalar birçok tasarım problemi çözümünde yoğun bir şekilde kullanılmaktadır. Ayrıca algoritmaların problem çözümlerinde göstermiş oldukları başarıların iyileştirilmesi, farklı türdeki problemlerde etkin çözüm arayışları gibi sebepler araştırmacıların konuya olan ilgisini sürekli arttırmaktadır. Bu amaçla daha genel ve başarılı yeni algoritma modelleri ortaya konulmakta, mevcut algoritmalar bazı operatörlerle iyileştirilmekte yada farklı algoritmaların bir arada kullanıldığı karma yapılar oluşturulmaktadır.

Popülasyon tabanlı sezgisel algoritmalar kaliteli çözümleri etkin bir şekilde bulmayı

hedeflerler. En iyi yada en iyiye yakın çözümleri makul bir süre içerisinde bulabilmeleri, deterministtik olmamaları, yerel en iyi tuzaklarından kurtulmak için çeşitli stratejilere sahip olmaları ve problemlere özgü olmamaları algoritmaların temel özellikleri olarak sıralanabilir [145, 151].

3.4. Tez Kapsamında Gerçekleştirilen Algoritmalar

Bu tez kapsamında popüler yada yeni önerilmiş bir sezgisel algoritma olmaları sebebi ile arı kolonilerinin davranışlarını örnek alan Yapay Arı Kolonisi (Artificial Bee Colony, ABC) [7], guguk kuşlarını modelleyen Guguk Kuşu Arama (Cuckoo Search, CS) [10], çözümler arasındaki farklılığa dayanan Diferansiyel Gelişim (Differential Evolution, DE) [8], ateş böceklerinin sosyal davranışlarını baz alan Ateş Böceği (Firefly, FF) [11], Newton'un evrensel kütle çekim kanunlarına dayanan Gravitasyonel Arama (Gravitational Search, GS) [12], kuşların yiyecek aramalarından esinlenen Parçacık Sürüsü Optimizasyon (Particle Swarm Optimization, PSO) [9] ve eğitim ve öğretim sisteminden esinlenerek geliştirilen Öğretme ve Öğrenme Temelli Optimizasyon (Teaching-Learning-Based Optimization, TLBO) [13, 14] algoritmaları gerçekleştirilmiştir.

Gerçekleştirilen algoritmaların tamamı popülasyon tabanlı sezgisel algoritmalardır ve bu algoritmalar rasgele oluşturdukları bir başlangıç popülasyonundaki bireylerin iteratif olarak birbirleri ile etkileşmelerini sağlayarak kesin çözüme ya da kesin çözümü yakın bir çözüm bulmaya çalışırlar [152]. Başlangıç popülasyonu oluşturulurken algoritmalarda kullanılacak her bir parametre kendisi için tanımlanan alt ve üst sınırlar içerisinde rasgele olarak üretilir. Popülasyondaki i'inci çözüm vektörünün gösterimi Eşitlik (3.3)'de, i'inci çözümün iyileştirilmesi için oluşturulan hız/fark vektörünün gösterimi Eşitlik (3.4)'de ve başlangıç popülasyonunun oluşturulma denklemi de Eşitlik (3.5)'de verilmiştir.

$$x_i = [x_{i_1}, x_{i_2}, ..., x_{i_D}] \tag{3.3}$$

$$v_i = [v_{i_1}, v_{i_2}, ..., v_{i_D}] \tag{3.4}$$

$$x_{ij} = x_j^{min} + rand()(x_j^{max} - x_j^{min}). \tag{3.5}$$

Eşitlik (3.5)'de i ve j indisleri sırası ile popülasyondaki birey numarasını ve optimize edilecek parametreyi göstermektedir. Gerçekleştirilen bu algoritmaların kendilerine özgü diğer özellikleri aşağıda açıklanmıştır. Formüllerde kullanılan NP popülasyon büyüklüğünü, D problem boyutunu, t iterasyon numarasını, rand(), (0,1) aralığında uniform dağılım fonksiyonunu, randn() ise (0,1) aralığında normal dağılım fonksiyonunu ifade etmektedir.

3.4.1. Yapay Arı Koloni Algoritması

Yapay arı kolonisi algoritması, arıların doğada yiyecek arama davranışlarını modelleyen popülasyon tabanlı sezgisel bir algoritmadır [7]. Bal arılarının kolonilerinde yapılacak işlere göre bir görev paylaşımı vardır. Bu görev paylaşımı yiyecek arama davranışı gösteren arıları üç gruba ayırır. Bunlar işçi arılar, gözcü arılar ve kaşif arılardır. İşçi arılar kendi hafızalarında bulunan kaynaklardan araştırma yaparken, gözcü arılar potansiyel olarak daha zengin kaynakları seçerek araştırma yaparlar. Kâşif arılar ise keşfedilmemiş rastgele kaynaklardan sorumludurlar. ABC algoritmasında popülasyon koloni ile ifade edilir. Yiyecek kaynağının yeri popülasyondaki çözüm vektörü iken kalitesi amaç fonksiyon değeridir. ABC algoritmasının temel adımları Algoritma 1'de gösterilmiştir.

Algorithm 1: ABC algoritmasının temel adımları

```
1: Başlangıç popülasyonunun oluşturulması
2: Uygunluk değerinin hesaplanması
3: repeat
4:     İşçi arı safhası;
5:     Gözcü arı safhası;
6:     Kâşif arı safhası;
7:     Seçme;
8:     Güncelleme;
9: until
```

İşçi arı aşaması, çözümlerin daha iyi arama alanı vaat eden bölgelere doğru hareket etmesini sağlar. Bu aşamada her bir çözüm vektörünün komşulukları Eşitlik (3.6) ile tanımlanan yeni çözüm üretme mekanizması kullanılarak aranır.

$$x_{ij}(t+1) = x_{ij}(t) + \phi_{ij}(x_{ij}(t) - x_{r_1 j}(t)) \tag{3.6}$$

Eşitlik (3.6)'da $x_{ij}(t+1)$ yeni aday çözüm vektörü, ϕ , $[-1, 1]$ aralığında üretilen rastgele bir değer, $r_1 \in [1, NP]$ aralığında $r_1 \neq i$ olmak şartıyla rastgele seçilmiş komşu bir çözüm vektörüdür.

Üretilen çözüm sonrası açgözlü seçim metodu kullanılarak, yeni çözüm ve mevcut çözüm karşılaştırılır. Yeni çözüm daha iyi ise mevcut çözümün yerini alır, aksi takdirde mevcut çözüm muhafaza edilir ve daha iyi bir çözüm üretilemediği için ilerleyici olmayan arama sayısının sayıldığı yerel arama sayacı bir arttırılır. Bu sayaç kâşif arılar için iyileştirilme yapılamayan kaynakların belirlenmesinde kullanılır.

Gözcü arı aşamasında da yeni çözüm üretmek için Eşitlik (3.6) kullanılır. İyileştirilme yapılacak çözümün belirlenmesinde iyi çözümlerin seçilme şansının daha fazla olduğu olasılıksal seçim metodu kullanılır. Böylelikle iyi bireylerin seçilme şansı artarak onların etrafında daha fazla yerel arama yapılabilmektedir. Temel ABC algoritmasında olasılık seçimi için Eşitlik (3.7) ile hesaplanan değerler kullanılır.

$$p_i = \frac{uygunluk_i}{\sum_{i=1}^{NP} uygunluk_i} \tag{3.7}$$

Burada p_i, i. çözümün seçilme olasılığına ve $uygunluk_i$, çözüm kalitesine (nektar miktarına) karşılık gelir.

Burada da işçi arılarda olduğu gibi açgözlü seçim metodu ile yeni çözümün mevcut çözümle değişip değişmeyeceğine karar verilir. Aynı şekilde, yerel arama sayısını tutan sayaçlar bu aşamada da güncellenir.

Doğada çalışma süreçleri nedeni ile işçi arılar ve gözcü arılar bazı gıda kaynaklarını tüketebilmektedir. Bu ABC algoritması açısından bir çözümün komşuluklarının yeterince aranması ve çözümün artık geliştirilemiyor olması anlamına gelir. Bu nedenle bu çözümün iyileştirilmeye çalışılmasına artık gerek yoktur, onun yerine rastgele başka bir çözüm üretilebilir. Kaynağın yeterince aranıp aranmadığı "limit" adı verilen algoritmaya özel bir kontrol parametresi ile belirlenir. x_i konumundaki

çözüm vektörü "limit" parametresi sayısınca gelişememiş ise x_i çözüm vektörü terk edilir ve o kaynağın arısı kaşif arı haline gelerek rastgele araştırma yapar.

3.4.2. Guguk Kuşu Arama Algoritması

Guguk kuşu arama algoritması 2009 yılında Yang ve Deb tarafından geliştirilmiştir [10]. Algoritma guguk kuşlarının yaşam tarzları ve üreme stratejilerinden esinlenmiştir. Guguk kuşları kendi yumurtalarını bazı farklı türlerdeki kuşların yuvalarına bırakabilme yeteneğine sahiptirler. Bu amaçla yeni yumurtlamış türleri seçebilmekte, mevcut yumurtayla kendi yumurtasını değiştirebilmektedir. Diğer taraftan ev sahibi kuşlarda farklı yumurtaları atarak ya da yuvalarını yeni yerlerde inşa ederek bu parazit davranışla mücadele etmeye çalışırlar. Algoritma genel olarak modellediği üreme davranışını idealize etmeyi amaçlamaktadır.

Algoritmada yuvadaki her bir yumurta bir çözümü temsil eder, guguk kuşu yumurtaları da yeni çözümleri temsil eder. Amaç yeni ve potansiyel olarak daha iyi çözümleri yuvadaki iyi olmayan yumurtalarla değiştirmektir. En basit formda her yuvada bir yumurta varken, her yuvada birden fazla yumurtanın olduğu karmaşık durumlarda gerçekleştirilebilir. CS algoritması aşağıda tanımlanan üç temel kurala dayanmaktadır.

1. Her guguk kuşu her seferde rastgele seçilen bir yuvaya yalnızca bir yumurta bırakır.
2. Yüksek kaliteli yumurtaların olduğu yuvalar bir sonraki nesle aktarılmaktadır.
3. Mevcut yuva sayısı sabittir ve guguk kuşu tarafından bırakılan yumurta bir olasılıkla diğer türler tarafından keşfedilir.

Algoritmada birinci aşamada rastgele seçilen yuvalar en iyi çözüm etrafında rastgele üretilen yeni çözümler ile yer değiştirir. Yeni çözüm üretme denklemi Eşitlik (3.8)'de verilmiştir.

$$x_i(t+1) = x_i(t) + \alpha.S.(x_i(t) - x_{best}(t)).rand() \tag{3.8}$$

Burada $\alpha > 0$ olmak şartıyla probleme göre ayarlanabilen adım büyüklüğüdür, S ise rastgele bir yürüyüşün uzunluğudur (Lévy flight) ve Eşitlik (3.9) ile hesaplanır.

$$S = \frac{u}{|v|^{1/\beta}} \tag{3.9}$$

Burada β [1, 2] ölçekleme faktörü, u ve v de $u = sigma.randn()$ ve $v = randn()$ olacak şekilde rastgele sayı üretecleridir.

İkinci aşamada ise, rastgele seçilen yuvalar rastgele seçilen diğer yuvalar ile pa mutasyon oranına bağlı olarak etkileşerek yeni çözüm üretir. Etkileşim olasılığı Eşitlik (3.10)'da verilmiştir.

$$P_{ij} = \begin{cases} 1 & \text{eğer } rand() < pa \\ 0 & \text{diğer} \end{cases} \tag{3.10}$$

Eşitlik (3.10)'da i ve j indisleri sırası ile popülasyondaki yuva ve değişken numaralarını göstermektedir. Üretilen bu yeni çözümler kalitelerine göre mevcut çözümlerle karşılaştırılır ve yer değiştirilir.

3.4.3. Diferansiyel Gelişim Algoritması

Popülasyon tabanlı sezgisel bir algoritma olan diferansiyel gelişim algoritması ilk olarak 2005 yılında geliştirilmiştir [8]. DE algoritması genetik algoritmaya benzetilebilir. Genetik algoritma çözüm vektörleri arasındaki benzerliklerden yararlanırken, DE algoritması çözüm vektörleri arasındaki farklılıklardan da yararlanmaktadır. DE algoritmasında popülasyon tabanlı bir algoritma olması dolayısı ile araştırma birçok noktadan yapılabilmektedir. Algoritma çalışma süresi boyunca operatörler yardımıyla popülasyonu iyileştirmeye çalışmaktadır. DE algoritmasında kullanılan operatörle mutasyon, çaprazlama ve seçim operatörleridir. DE algoritmasının temel adımları Algoritma 2'de gösterilmiştir.

Algorithm 2: DE algoritmasının temel adımları

1: Başlangıç popülasyonunun oluşturulması
2: Uygunluk değerinin hesaplanması
3: **repeat**
4: Mutasyon;
5: Çaprazlama;
6: Seçme;
7: Güncelleme;
8: **until**

Mutasyon, mevcut vektörün bazı parametre değerlerinin rastgele üretilmiş bir değerde değiştirilmesidir. DE algoritmasında farklı mutasyon operatörleri bulunmaktadır. En yaygın kullanılan mutasyon operatörlerinden biri DE/rand/1 dir. Bu operatörde değişikliğe uğrayacak çözüm vektörü dışında, popülasyondan üç farklı çözüm vektörü seçilerek bunlardan ikisinin farklı alınıp ölçeklenerek diğer vektöre eklenmesi ile yeni fark vektörü üretilir. DE/rand/1 mutasyon operatörü Eşitlik (3.11)'de verilmiştir.

$$v_i(t) = x_{r_1}(t) + F(x_{r_2}(t) - x_{r_3}(t)) \tag{3.11}$$

Burada r_1, r_2, r_3, $[1, NP]$ aralığında her bir iterasyonda rastgele atanan birbirinden farklı tamsayılardır. F genellikle $[0, 1]$ arasında değer alan ölçekleme faktörü, $v_i(t)$ ise t iterasyona ait fark vektörüdür.

Çaprazlama operatöründe mutasyon sonrası oluşan $v_i(t)$ fark vektörü ile değişikliğe uğrayacak $x_i(t)$ vektörü kullanılarak yeni aday çözüm vektörü oluşturulur. Çaprazlama operatörü Eşitlik (3.12)'de verilmiştir.

$$u_i(t) = \begin{cases} v_i(t) & \text{eğer } rand() \leq CR \\ x_i(t) & \text{diğer} \end{cases} \tag{3.12}$$

Burada $u_i(t)$ aday çözüm vektörü, CR çaprazlama oranıdır. Her parametre için üretilen $(0, 1)$ aralığındaki rastgele sayı CR'den düşükse aday çözümün o parametre değeri fark vektöründen, değilse $x_i(t)$ çözümünden alınır.

Seçme operatöründe aday çözüm vektörü ile değişikliğe uğrayacak vektörün hangisinin yeni popülasyonda yer alacağı açgözlü seçim metodu kullanılarak yani

uygunluk fonksiyonu değerleri karşılaştırılarak karar verilir. Seçme operatörü Eşitlik (3.13)'de verilmiştir.

$$x_i(t+1) = \begin{cases} u_i(t) & \text{eğer } f\left(u_i(t)\right) \leq f\left(x_i(t)\right) \\ x_i(t) & \text{diğer} \end{cases} \tag{3.13}$$

Burada $x_i(t+1)$ popülasyondaki yeni birey, $f(u_i(t))$ ve $f(x_i(t))$ sırası ile aday çözüm vektörü ile değişikliğe uğrayacak vektörün uygunluk fonksiyon değerleridir.

3.4.4. Ateş Böceği Algoritması

Ateş böceği algoritması, ateşböceklerinin sosyal davranışlarından esinlenerek Yang tarafından geliştirilmiş bir optimizasyon algoritmasıdır [11]. Bu algoritmanın diğer popülasyon tabanlı algoritmalarla bir çok ortak özelliği bulunmaktadır. Algoritmada ateş böceği sürüsü popülasyon olarak düşünülmektedir. Algoritma ateş böceklerinin ışıklarını yakıp söndürmesi ve bu sayede diğer ateş böceklerini kendisine çekmesi prensibine dayanmaktadır.

Algoritmada verilen bir optimizasyon probleminin çözümünde, popülasyonun parlak ve daha çekici yerlere gitmesi çözüm kalitesi olarak tanımlanan yanıp sönen ışıklar ya da ışık şiddeti ile ilişkilidir. Popülasyondaki tüm bireyler çözüm kaliteleri ve mesafeleri oranında birbirlerini çekmektedir. Yani ateş böceği ne kadar parlak olursa çözüm kalitesi o kadar iyidir ve diğer ateş böcekleri için o kadar çekicidir. Bununla beraber mesafe, parlaklığı azalttığı için çekim kuvvetini de azaltmaktadır. FF algoritmasının temel adımları Algoritma 3 ile gösterilmiştir.

Algorithm 3: FF algoritmasının temel adımları

1: Başlangıç popülasyonunun oluşturulması
2: Uygunluk değerinin hesaplanması (ışık şiddeti)
3: Işığın emilim katsayısının belirlenmesi (γ)
4: **repeat**
5: Her bir adayın kendinden daha iyi çözümler tarafından etkilendiği yeni çözüm üretilir;
6: Uygunluk değerinin hesaplanması
7: Güncelleme;
8: Popülasyon sıralaması ve en iyi çözümün belirlenmesi;
9: **until**

Ateşböceği algoritmasında iki önemli nokta vardır. Bunlar ışık yoğunluğunun değişimi ve popülasyondaki bireylerin çekicilik formülüdür. Algoritmada ateş böceğinin çekiciliğinin parlaklığı tarafından belirlendiği varsayılır. Buda popülasyondaki bireyin amaç fonksiyon değeri ile ilişkilidir.

Bir optimizasyon problemi için parlaklık $I(x) \infty f(x)$ şeklinde gösterilebilir. Bunun yanında çekicilik değeri (β) görecelidir ve ateş böceği i ile ateş böceği j arasındaki mesafe (r_{ij}) değerine bağlı olarak değişecektir. Dolayısı ile ışık kaynağından uzaklaştıkça ışık emilerek şiddeti azalacaktır. En basit haliyle ışık şiddeti mesafenin karesine bağlı olarak Eşitlik (3.14)'deki gibi formülüze edilebilir.

$$I(r) = \frac{I_s}{r^2} \tag{3.14}$$

Burada r mesafe, I_s de ışık şiddetidir. Herhangi iki ateş böceği arasındaki mesafe Eşitlik (3.15) ile yeni çözüm üretme denklemi de Eşitlik (3.16) ile hesaplanır.

$$r_{ik} = \sqrt{\sum_{j=1}^{D} (x_{i,j} - x_{k,j})^2} \tag{3.15}$$

$$x_i(t+1) = x_i(t) + \beta_o e^{-\gamma r_{ik}^2}(x_k(t) - x_i(t)) + \alpha \varepsilon_i \tag{3.16}$$

Burada $k \in [1, NP]$ aralığında $k \neq i$ olmak şartıyla komşu bir çözüm vektörü, α adım büyüklüğü, ε_i Gauss veya uniform dağılımlı bir vektör, γ ateş böcekleri arasında çekicilik değişimlerini belirleyen bir sabit, r ateş böcekleri arasındaki mesafe ve β_o'da onların çekiciliğini göstermektedir.

3.4.5. Gravitasyonel Arama Algoritması

Gravitasyonel arama algoritması, Newton'un evrensel kütle çekim kanunlarına dayanarak geliştirilmiş popülasyon tabanlı bir optimizasyon algoritmasıdır [12]. Newton yerçekimi kanununda, her bir parça diğer her parçayı belirli bir güçle kendine doğru çekmektedir. Buna "yerçekimsel güç" adı verilir. GS algoritması bu temel

prensip üzerinden esinlenmiştir. Algoritmada kütleler olarak adlandırılan bir dizi ajan yerçekimsel gücün simülasyonu ile optimum çözümü bulmaya çalışır.

Buna göre G yerçekimi sabiti olmak üzere, aralarındaki uzaklık R olan ve kütleleri sırası ile M_1 ve M_2 olan iki cisim arasındaki yerçekimsel güç denklemi Eşitlik (3.17)'de verilmektedir.

$$F = G\frac{M_1 M_2}{R^2} \tag{3.17}$$

GS'nin hız vektörü ve yeni çözüm üretme denklemi aşağıda verilen Eşitlik (3.18) ve (3.19) ile güncellenir.

$$v_i(t+1) = rand()v_i(t) + \alpha_i(t) \tag{3.18}$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{3.19}$$

Burada $\alpha_i(t)$, hareket yasası gereği i'inci çözümün t anındaki hız değişimidir ve Eşitlik (3.20) ile hesaplanır.

$$a_i(t) = \frac{F_i(t)}{M_{ii}(t)} \tag{3.20}$$

Burada $M_{ii}(t)$ eylemsizlik kütlesi, $F_i(t)$'de Eşitlik (3.21) ile hesaplanan diğer bireyler tarafından uygulanan rastgele ağırlıklandırılmış çekim etkilerinin toplamıdır.

$$F_i(t) = \sum_{j=1, j\neq i}^{NP} rand()F_{ij}(t) \tag{3.21}$$

$F_{ij}(t)$ değeri ise belirli bir t anında i'inci bireyin j'inci bireye etki eden çekim kuvvetidir ve Eşitlik (3.22) ile hesaplanır.

$$F_{ij}(t) = G(t)\frac{M_{pi}(t) x M_{aj}(t)}{R_{ij}(t) + \varepsilon}(x_j(t) - x_i(t)) \tag{3.22}$$

Bu denklemde, M_{aj} j'inci bireyin aktif yerçekimi kütlesi, M_{pi}, i'inci bireyin pasif yerçekimi kuvveti, $G(t)$, t anındaki yerçekimi sabiti, ε küçük bir sabit, $R_{ij}(t)$ iki birey arasındaki Öklid uzaklığıdır. Yerçekimi ve eylemsizlik kütleleri bireylerin uygunluk değerlerin bağlı olarak hesaplanır. Ağır bir kütle daha etkili bir çözüm anlamına gelir. İyi bireyler yüksek çekim kuvvetine sahiptirler ve yavaş değişim gösterirler. Algoritmada yerçekimi ve eylemsizlik kütleleri bir birine eşit varsayılır ve aşağıdaki Eşitlik (3.23) ile hesaplanır. ($M_{ai} = M_{pi} = M_{ii} = M_i$)

$$M_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)} (\sum_{j=1}^{N} m_j(t))^{-1} \tag{3.23}$$

Burada $fit_i(t)$ değeri, i'inci nesnenin t anındaki amaç fonksiyonu değeridir. $best(t)$ değeri, tüm nesnelerin elde ettiği amaç fonksiyonu arasında en iyi değeri ifade ederken $worst(t)$ değeri, elde edilen amaç fonksiyon değerleri arasında en kötü değeri göstermektedir. GS algoritmasının temel adımları Algoritma 4 ile gösterilmiştir.

Algorithm 4: GS algoritmasının temel adımları

```
1: Başlangıç popülasyonunun oluşturulması
2: repeat
3:    Uygunluk değerinin hesaplanması;
4:    En iyi ve en kötü çözümlerin belirlenmesi;
5:    Yerçekimi sabitinin güncellenmesi
6:    Her bir birey için M ve α değerlerinin hesaplanması;
7:    Hız vektörünün oluşturulması;
8:    Yeni aday çözüm üretilmesi;
9:    Güncelleme;
10: until
```

3.4.6. Parçacık Sürüsü Optimizasyon Algoritması

Parçacık sürüsü optimizasyon algoritması popülasyon tabanlı sezgisel bir algoritmadır. Algoritmada problem çözümü, kuşların yiyecek kaynaklarına doğru yaptıkları davranışlara benzetilmektedir. Bu amaçla kuş sürülerinin iki boyutlu hareketlerinden esinlenerek Kennedy ve Eberhart tarafından geliştirilmiştir [9]. Algoritmada problem için her bir aday çözüm parçacıkla popülasyon ise sürü ile ifade edilir. Diğer evrimsel algoritmalarda olduğu gibi PSO'da da aday çözümlerden oluşan başlangıç popülasyonu rastgele oluşturulur ve her bir iterasyonda bu çözümler

iyileştirilmeye çalışılır. Aday çözümlerin iyileştirilme işlemi ise popülasyonda o andaki en iyi çözüm (gbest) ve kendi kişisel en iyi çözümü (pbest) kullanılarak olur. Bu sayede bireyler arasında bilgi paylaşımı olmaktadır. PSO algoritmasının temel adımları Algoritma 5'de gösterilmiştir.

Algorithm 5: PSO algoritmasının temel adımları

```
1: Başlangıç popülasyonunun oluşturulması
2: Uygunluk değerinin hesaplanması
3: repeat
4:     En iyi çözümün belirlenmesi (gbest);
5:     Kendi en iyi değerinin güncellenmesi (pbest);
6:     Hız vektörünün oluşturulması;
7:     Yeni aday çözüm üretilmesi;
8:     Güncelleme;
9: until
```

Algoritmada kendi en iyi değeri ve popülasyondaki en iyi değerin bulunmasından sonra hız vektörü ve yeni çözüm vektörü aşağıda verilen Eşitlik (3.24) ve (3.25) ile güncellenir.

$$v_i(t+1) = w * v_i(t) + c_1 rand()(x_i^{best} - x_i(t)) + c_2 rand()(x_{gbest} - x_i(t)) \quad (3.24)$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (3.25)$$

Burada, w bir önceki hız değerinin etkisini belirleyen ağırlık, c_1 ve c_2 sosyal ve bilişsel öğrenme faktörleridir. c_1 ve c_2, her parçacığı pbest ve gbest pozisyonlarına doğru çeken, hızlanma terimlerini ifade eden sabitlerdir. c_1, parçacığın kendi bilgilerine göre hareket etmesini, c_2 ise sürüdeki diğer parçacıkların bilgilerine göre hareket etmesini sağlar. Bu değerlerin düşük seçilmesi araştırma uzayının çok yavaş aranmasına sebep olurken, yüksek seçilmesi, araştırma uzayının hızlı bir şekilde taranması sağlayarak hedefe ulaşmayı hızlandırırken, optimum bölgenin es geçilmesine sebep olabilir.

3.4.7. Öğretme ve Öğrenme Temelli Optimizasyon Algoritması

Öğretme ve öğrenme temelli optimizasyon algoritması 2011 yılında Rao tarafından geliştirilmiştir [13]. Eğitim ve öğretim sisteminden esinlenerek geliştirilen algoritmadan farklı tipteki optimizasyon problemlerine etkili çözümler bulması beklenmektedir. TLBO algoritması, doğal yaşamı modelleyen diğer optimizasyon algoritmalarında olduğu gibi, küresel çözümü bulmak için mevcut çözümler üzerinde iyileştirmeler yapan popülasyon tabanlı bir yöntemdir. TLBO algoritmasında popülasyon öğrencilerden oluşan bir grup yada sınıf olarak kabul edilir. Algoritmada her bir öğrenci tasarım değişkenlerini oluştururken, öğrencilerin başarıları da amaç fonksiyon değerlerini temsil etmektedir. Öğretmen ise elde edilen en iyi çözüm olarak kabul edilir.

Algoritmada temelde öğretmenin sınıftaki öğrencilere etkisi ve sınıf bireylerinin birbirlerine olan etkisi esas alınmaktadır. Öğretmen genellikle kendi bilgilerini öğrencilerle paylaşarak onların kalitesini arttırmaya çalışır. Sınıf içerisindeki bireylerin birbirleri ile etkileşimi ile de kaliteli bireylerin kendilerine göre daha az kaliteli bireyleri etkileyerek toplam kalitenin artışı sağlanır. Bu bağlamda TLBO algoritmasının işleyişini iki farklı aşamaya ayırmamız mümkündür, bunlar öğretmenin öğrencilere olan etkisi *öğretmen aşaması* ve öğrencilerin birbirlerine olan etkisi *öğrenci aşaması* dır. Öğretmen aşaması öğretmenden öğrenme anlamına gelirken, öğrenci aşaması öğrencilerin kendi aralarındaki etkileşim yoluyla öğrenmesi anlamına gelir.

Öğretmen aşamasında, toplumda en bilgili kişi öğretmen olarak kabul edilir, bu nedenle algoritmada en iyi öğrenci bir öğretmen olarak taklit edilir. Öğretmen kendi kapasitesine göre öğrencileri kendi seviyesine doğru çekerek ortalama kaliteyi arttırmak için çalışır. Ortalama iyileştikçe, daha üstün kalitede yeni bir öğretmen üretilmesi beklenir. Bu nedenle popülasyondaki öğretmen değişebilir. Teoride, öğretmen öğrencilerinin eğitimi için azami çaba ortaya koyarak onları kendi seviyesine getirmeye çalışacaktır. Fakat pratikte bu mümkün değildir, öğretmen sadece mevcut öğrencilerin kalitesi ve öğretimin şeklinin kalitesine bağlı olarak sınıf ortalamasını bir dereceye kadar yukarı taşıyabilir. Bu birçok faktöre bağlı olarak rastgele bir süreç izler. i iterasyon sayısı olmak üzere M_i ve T_i sırası ile her bir iterasyondaki ortalama ve öğretmen değişkenleridir. T_i, ortalama değer M_i'yi

kendi seviyesine doğru taşımaya çalışacaktır. Bundan dolayı yeni ortalama değer M_{yeni} öğretmen T_i ye bağlı olarak belirlenecektir. Popülasyondaki bireyler, mevcut ortalama değer ve yeni ortalama değer arasındaki farka bakılan Eşitlik (3.26)'deki formül ile güncellenecektir.

$$ortalama\_fark_i = rand()(T_i - T_F M_i) \tag{3.26}$$

Burada $rand()$, $[0, 1]$ aralığında değer alan düzgün dağılımdan gelen gerçel rastgele bir değerdir. T_F ise 1 veya 2 değerlerinden herhangi birini Eşitlik (3.27)'deki formül ile rastgele alan öğretme faktörüdür.

$$T_F = round[1 + rand()\{2 - 1\}] \tag{3.27}$$

Elde edilen ortalama fark değeri kullanılarak mevcut çözümler Eşitlik (3.28) ile iyileştirilmeye çalışılır.

$$x_i(t+1) = x_i(t) + ortalama\_fark_i \tag{3.28}$$

Öğrencilerin öğretmenden alınan bilgilerin dışında kendi aralarındaki etkileşim yoluyla bilgilerini arttırmaları öğrenci aşaması olarak kabul edilir. Öğrenciler rastgele olarak birbirleri ile grup çalışmaları, sunumlar, yardımlaşma ve iletişim gibi araçlarla birbirlerinden etkilenirler. Bu çalışmalarda öğrenciler kendilerinden daha fazla bilgili diğer öğrencilerden bilgi edinirler. Algoritmada öğrenciler arası etkileşim, temelde iyi öğrencilerin kötü öğrencileri etkilemesi prensibi üzerine kurgulanmıştır. Öğrenci aşaması aşağıdaki şekilde kodlanabilir:

$$\begin{aligned}
&if\ f(x_i(t)) < f(x_{r_1}(t)) \\
&\quad x_i(t+1) = x_i(t) + rand()(x_i(t) - x_{r_1}(t)) \\
&else \\
&\quad x_i(t+1) = x_i(t) + rand()(x_{r_1}(t) - x_i(t)) \\
&end\ if
\end{aligned} \tag{3.29}$$

Burada r_1, $i \neq r_1$ olmak üzere, sınıftan rastgele seçilmiş başka bir öğrencidir. Elde

edilen yeni çözümler mevcut çözümlerden daha iyi ise kabul edilir, aksi takdirde eski çözümler değişmeden muhafaza edilir.

3.4.8. Gerçekleştirilen Algoritmaların Karşılaştırılması

Bu bölümde gerçekleştirilen algoritmaların sahip oldukları özellikler kısaca karşılaştırılmaktadır. Yapılan çalışmada algoritmaların temel modelleri gerçekleştirildiği için karşılaştırmada bu en temel modeller üzerinden yapılmıştır.

Algoritmalar genellikle olasılık tabanlıdır ve bazı kontrol parametreleri gerektirir [145, 151]. Popülasyon büyüklüğü ve iterasyon sayısı popülasyon tabanlı tüm sezgisel algoritmaların ortak parametreleridir. Bunlara ek olarak algoritmalar karakteristiklerine göre bazı özel kontrol parametreleri bulundurmaktadır. Bu kontrol parametrelerinin uygun değerlere ayarlanması; algoritmaların performanslarını önemli derecede etkiledikleri için oldukça önemlidir. Ayrıca parametre sayısının fazlalığı, ayarlanmaları için ihtiyaç duyulan deney sayısını arttırmakta dolayısı ile bir dezavantaj oluşturmaktadır. Gerçekleştirilen algoritmalardan TLBO algoritması hiç bir özel kontrol parametresi bulundurmazken, ABC algoritması bir kontrol parametresi diğerleri de iki ya da daha fazla özel kontrol parametresi bulundurmaktadır.

Popülasyon tabanlı sezgisel algoritmalar iteratif olarak popülasyonun kalitesini arttırmayı çalışırlar [145]. Bu amaçla ABC, CS, DE, FF, GS ve TLBO algoritmaları yeni üretilen aday çözümleri mevcut çözümlerle kıyaslayarak daha iyi olanın muhafaza edildiği açgözlü seçim metodu kullanırken, PSO algoritması yeni üretilen aday çözümü herhangi bir karşılaştırma yapmadan mevcut çözümle yer değiştirir.

Gerçekleştirilen algoritmaların çalışma süreçleri incelendiğinde; ABC algoritması yeni çözüm üretmek için mevcut çözüm setinden sadece bir parametre değiştirirken, DE algoritması çaprazlama oranına bağlı olarak parametre değiştirir, CS, FF, GS, PSO ve TLBO algoritmaları ise çözüm setindeki tüm parametreleri değiştirir.

Genel olarak popülasyon tabanlı sezgisel algoritmalar yeni çözüm üretirken popülasyondaki diğer çözümlerle etkileşim içerisindedir. Örneğin ABC algoritması,

popülasyondan rastgele seçilmiş farklı bir çözüm seti ile etkileşir, DE algoritması tercih edilen stratejiye bağlı olarak en az iki en fazla beş farklı çözüm seti ile etkileşir, FF algoritması popülasyondaki kendisinden daha kaliteli çözüm setleri ile etkileşir, GS algoritması ise popülasyondaki tüm diğer çözüm setleri ile etkileşir.

Bunun yanında CS, PSO ve TLBO algoritmaları yeni çözüm üretirken popülasyondaki en iyi çözüm setini direk olarak kullanır, DE algoritması tercih edilen stratejiye göre faydalanırken diğerleri en iyi çözüm setini direk olarak kullanmazlar.

Algoritmaların hesaplama karmaşıklıkları, çalışma süreçleri içerisinde kullanıldıkları matematiksel formüllere göre değişmektedir. ABC, CS, DE, PSO ve TLBO algoritmaları basit matematiksel formüller kullanırken FF ve GS algoritmaları yeni çözüm üretirken Öklid uzaklık formülünü kullanmaktadırlar. Bundan dolayı FF ve GS algoritmalarının çalışma süreleri daha uzun sürmekte ve hesaplama karmaşıklık değerleri diğerlerine göre çok daha yüksek çıkmaktadır.

4. BÖLÜM

ALGORİTMALAR İÇİN PARALELLEŞTİRME MODELLERİ

4.1. Giriş

Gelişen teknoloji ile beraber artan hesaplama ihtiyaçları paralel donanım ve yazılımların sağladığı performans artışı ile aşılmaya çalışılmaktadır. Sezgisel algoritmaların birçoğunun doğası gereği paralelleştirilebilir yapıda olması, araştırmacıların paralel teknikleri geliştirmelerine ve bu teknikleri kullanarak daha verimli uygulamalar gerçekleştirmelerine neden olmaktadır. Algoritmaların paralelleştirilmesiyle hesaplama dağıtılarak zamandan kazanç sağlanır ve çözüm kalitesinin arttırılması hedeflenir. Tabii ki gerçekleştirilen algoritmanın ve paralelleştirme modelinin farklı avantaj ve dezavantajları olabilir.

Bu tez kapsamında genel olarak popülasyon tabanlı sezgisel algoritmalar üzerinde çalışılmıştır. Popülasyon tabanlı sezgisel algoritmalarının paralelleştirilmesinde kullanılan temel modeller master-slave, fine-grained, coarse-grained (sub-population) ve hybrid paralelleştirme modelleridir [153, 154]. Bu modeller donanım ihtiyaçları ve verimlilik bakımından farklılık arz ederler.

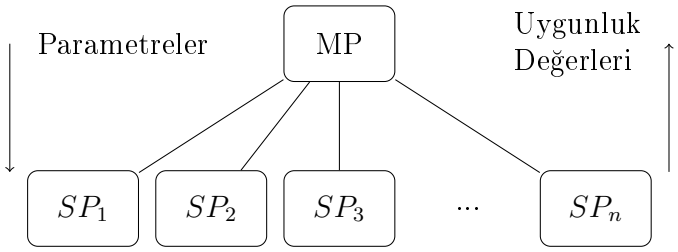
Birbiri ile veri haberleşmesi yapan bu temel modellerin dışında farklı işlemciler veya makinalar üzerinde birbirinden bağımsız çalışan problem çözümlerinden de bahsedilebilir. İşlemciler arasında hiçbir iletişimin olmadığı bu yöntem eşzamanlı çalışmalarda yada çok sayıda koşmanın gerektiği durumlarda faydalı olabilir. Bu yöntem genellikle algoritmaların farklı parametre setlerinin problemin çözümüne etkisinin incelenmesi esnasında zaman kazanmak için kullanılır. Bu şekilde gerçekleştirilen paralellik sezgisel algoritmalara bir kazanç sağlamaz.

Sezgisel algoritmaların paralelleştirilmesinde kullanılan ilk model 1976'da Bethke tarafından standart genetik algoritmaya uyarlanan master-slave modelidir [66]. Ayrıca, Tanese [83] , Pettey ve arkadaşlarının [84] yaptığı çalışmalar en eski paralel uygulamalardan ikisidir ve çalışmalarda standart genetik algoritmanın popülasyonu küçük parçalara bölünerek coarse-grained paralel modeli gerçekleştirilmiştir. Günümüzde ise bilgisayar mimarilerindeki çok çeşitli gelişmeler (paylaşımlı bellek, kümeler (cluster), ızgaralar (grid), ekran kartları, vb) paralel sezgisel algoritmaların uygulama alanlarının genişlemesine sebep olmaktadır.

4.2. Master-Slave Modeli

Master-slave modeli her bir aday çözümün uygunluk değerinin farklı bir işlemci üzerinde eşzamanlı olarak hesaplandığı bir yaklaşımdır. Master işlemci popülasyonu yönetir, slave işlemcilerde hesaplanan uygunluk değerlerini toplar ve algoritmanın yeni nesil üretme operatörlerini gerçekleştirir [153, 155]. Ayrıca üretilen bu yeni bireylerin uygunluk değerlerinin hesaplanması için tekrar slave işlemcilere gönderir. Slave işlemciler ise sadece aday çözümlerin uygunluk değerlerini hesaplar. Bireylerin uygunluk değerlerinin popülasyondaki diğer bireylerden bağımsız olması sebebi ile de slave işlemciler arasında iletişim gerçekleşmez. Eğer popülasyondaki birey sayısı slave işlemci sayısından fazla ise, bireyler işlemcilere mümkün olduğu kadar eşit sayıda dağıtılmaya çalışılır. Bu model hem paylaşımlı bellek hem de dağıtık bellek makinalarında gerçekleştirilebilir.

Master-slave modeli senkron ve asenkron olmak üzere iki farklı şekilde uygulanabilmektedir [155]. Senkron yaklaşımında master işlemci uygunluk değerlerinin hesaplanması için aday çözümleri slave işlemcilere gönderir, tüm hesaplamalar tamamlandıktan sonra seçim ve değiştirme işlemi gerçekleştirilir. Buna karşılık asenkron yaklaşımında seçim ve değiştirme işlemi için popülasyondaki tüm bireylerin uygunluk değerlerinin hesaplanması beklenmez. Yeni değerlendirilen bireyin popülasyona eklenebilmesi için direk seçim ve değiştirme işlemi yapılır. Senkron ve asenkron yaklaşımlar arasındaki bazı algoritmik değişiklikler nedeniyle farklı sonuçlar üretilebilir.



Şekil 4.1. Master-Slave Modeli

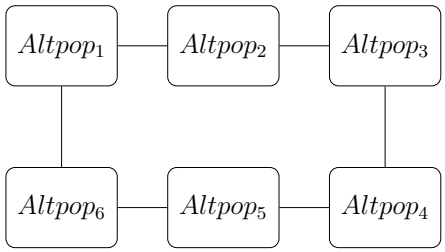
Birçok gerçek dünya probleminde, aday çözümlerin uygunluk değerlerinin hesaplaması algoritmanın en zaman alıcı adımıdır ve genel olarak bu hesaplama zamanı her bir aday çözüm için aynı olduğu varsayılır. Master-slave modeli özellikle uygunluk değerlerinin hesaplanması için harcanan zamanın paralel modellerdeki işlemciler arasındaki iletişim için harcanan zamana kıyasla çok daha büyük olduğu problemlerin çözümlerini hızlandırmak için tercih edilir. Bu modelle elde edilebilecek hızlanma, en iyi durumda kullanılan slave işlemci sayısı ile doğrusal olarak orantılı olacaktır. Ayrıca bu modelin popülasyon tabanlı sezgisel algoritmaların paralelleştirilmesi için uygulanacak en kolay yaklaşım olduğu da söylenebilir. Şekil 4.1'de modelin şematik olarak gösterimi ve Algoritma 6'da algoritmik açıklaması verilmiştir.

Algorithm 6: Master-slave modeline dayalı evrimsel algoritma

```
başlangıç popülasyonunun oluşturulması
for all birey do in parallel
    uygunluk değerinin hesaplanması ;
end for
while durdurma kriteri sağlanmıyorsa do
    seçme;
    yeni birey üretme;
    mutasyon;
    for all birey do in parallel
        uygunluk değerinin hesaplanması;
    end for
    yeni popülasyona iyi bireylerin eklenmesi, kötü bireylerin çıkarılması;
end while
```

4.3. Coarse-Grained Modeli

Coarse-grained modelinde popülasyon alt popülasyonlara bölünerek birbirleri ile bağlantılı yarı bağımsız aday çözüm grupları oluşturulur [153]. Bu aday çözüm



Şekil 4.2. Coarse-Grained Modeli

gruplarının her birine ada denilmektedir. Oluşturulan bu adaların her biri farklı bir işlemcide algoritmaların seri kodlarını çalıştırır ve çeşitli kriterlere göre bu adalar arası iletişim gerçekleştirilir. Bu sayede alt popülasyonlar arası bilgi alışverişi sağlanmış olur.

Sezgisel algoritmalarda ki coarse-grained modeli basit bir paralelleştirme değildir. Paralelleştirmenin yanında algoritmaların çalışma biçimi de değişmektedir. Genel olarak bu model ile popülasyondaki çeşitlilik sağlanarak algoritmanın yerel optimuma takılması engellenmeye çalışılmaktadır. Bunun yanı sıra her bir alt popülasyonların arama uzayında farklı bölgeleri arayacağı umulmaktadır. Bu nedenle, daha iyi ve daha çeşitli çözümler elde edilebilmektedir. Ayrıca bu modelde işlemciler arası iletişim diğer modellere göre daha azdır. Şekil 4.2'de modelin şematik olarak gösterimi ve Algoritma 7'de algoritmik açıklaması verilmiştir.

Bu modelde alt popülasyonlar arası iletişim kriterlerinin belirlenmesi amacı ile farklı parametreler ortaya çıkmaktadır. Bu parametreler: göç topolojisi, göç stratejileri, göç sıklığı ve göç eden birey sayısıdır [153, 155]. Genel olarak algoritmaların performansı, yakınsama hızları ve çalışma zamanları bu parametre değerlerine bağlı olarak değişmektedir [156, 157]. Şu anda bu parametrelerin ideal değerlerini belirleyecek kesin bir formül bulunmamasına rağmen yapılan çeşitli teorik ve pratik çalışmalar yaklaşık değerler sunmaktadır. Alt popülasyonlar arasında bireylerin göç kriterleri ile ilgili farklı parametreler aşağıdaki dört alt başlıkta açıklanmıştır.

Algorithm 7: Coarse-grained modeline dayalı evrimsel algoritma

```
1: N bireyden oluşan P adet alt popülasyonun oluşturulması, iterasyon := 1;
2: while durdurma kriteri sağlanmıyorsa do
3:    for all alt popülasyon do in parallel
4:       uygunluk değerinin hesaplanması;
5:       seçme;
6:       if iletişim frekansı şartı sağlanıyorsa then
7:          komşu alt popülasyona K adet bireyin gönderilmesi (K<N);
8:          komşu alt popülasyondan K adet bireyin alınması;
9:          alt popülasyondaki K adet bireyin yer değiştirmesi;
10:      end if
11:      yeni birey üretme;
12:      mutasyon;
13:   end for
14:   iterasyon := iterasyon + 1;
15: end while
```

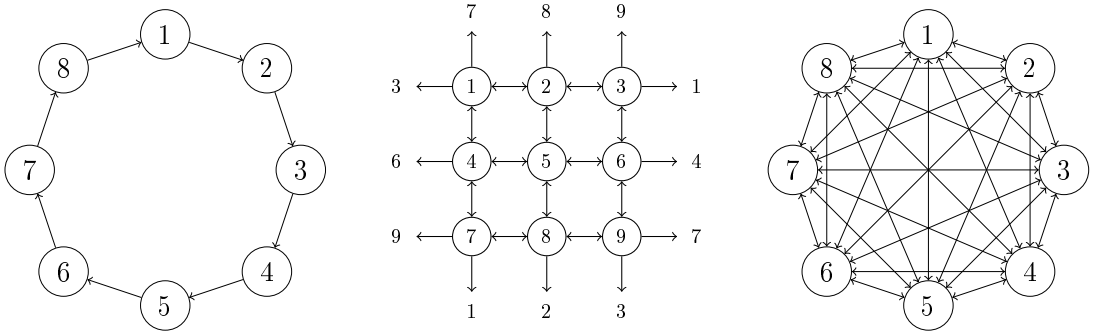
4.3.1. Göç Topolojileri

Adaların düzenlenmiş olduğu topoloji, ağ üzerinki işlemcilerin bağlantılarını tarif eden önemli bir parametredir. Bu parametre aynı zamanda algoritmaların çalışma hızını etkilemektedir [158]. Bilimsel çalışmalarda en yaygın kullanılan topolojiler: halka, rastgele, grid ve tam bağlantılı topolojilerdir. Topolojiler her düğümün bir ada olduğu ve adalar arası göç yönünün oklarla gösterildiği grafiklerle temsil edilebilir. Şekil 4.3'de yaygın kullanılan bazı topolojilerin grafiksel gösterimi verilmiştir.

Halka topolojisinde her ada bir önceki adadan bireyler alır ve bir sonraki adaya bireyler verir (Şekil 4.3(a)). Grid topolojisinde adalar bir ızgaraya yerleştirilmiş şekildedir ve her bir ada kuzey, güney, doğu ve batı yönündeki adalarla iletişim kurar (Şekil 4.3(b)). Bu topolojide ızgaranın üst ve alt kenarı ile sağ ve sol kenarları birbirine bağlanmış bir yumak şeklinde düşünülür. Tam bağlantılı topolojide adından da anlaşılacağı üzere her ada diğer tüm adalarla bağlantılıdır (Şekil 4.3(c)). Rastgele topolojide ise, adalar arasındaki göç bağlantısı rastgele olarak gerçekleşir.

4.3.2. Göç Stratejileri

Göç stratejileri iki kısımdan oluştur. Birinci kısım başka bir adaya göç edecek



Şekil 4.3. Göç topolojileri, a)Halka, b)Grid ve c)Tam bağlantılı

bireyin seçimidir. İkinci kısım başka bir adadan gelen yeni bireyin hangi bireyle yer değiştireceğidir. Göç edecek bireyin belirlenmesinde en yaygın kullanılan metot en iyi bireyin ya da rastgele bir bireyin seçilmesidir. Yer değiştirilecek bireyin belirlenmesinde ise en yaygın kullanılan metot en kötü birey ya da rastgele bir bireyin seçilmesidir. Bu durumda aşağıda tanımlanan dört farklı göç stratejisi ortaya çıkmaktadır [156]. Bu stratejilerin dışında daha farklı göç stratejileride oluşturmak mümkündür.

- K adet en iyi birey seç, K adet en kötü birey ile yer değiştir.
- K adet en iyi birey seç, K adet rastgele seçilmiş birey ile yer değiştir.
- K adet rastgele birey seç, K adet en kötü birey ile yer değiştir.
- K adet rastgele birey seç, K adet rastgele seçilmiş birey ile yer değiştir.

4.3.3. Göç Sıklığı

Adalar arasında bilgi transferleri için göç işlemi olmak zorundadır. Göç işlemi ya senkron bir şekilde her n inci iterasyonda ya da periyodik olmayan zamanlarda asenkron bir şekilde gerçekleşebilir. Her iki durumda da göç sıklığı algoritmaların davranışlarını değiştirmektedir. Genellikle daha sık bir göç işleminin daha hızlı bir yakınsamaya yol açtığı kabul edilmektedir [156, 157]. Ancak aşırı göç sıklığı da algoritmaların yerel optimuma takılma olasılığını arttırmaktadır [158].

Bunun yanında göç sıklığı algoritmaların çalışma zamanlarını da etkilemektedir.

Seçme ve değiştirme stratejileri, alt popülasyon boyutu ve göç büyüklüğüne bağlı olarak göç işlemi fazlaca zaman alabilir. Bu nedenle daha düşük bir göç sıklığı daha yüksek bir paralellik anlamına gelir.

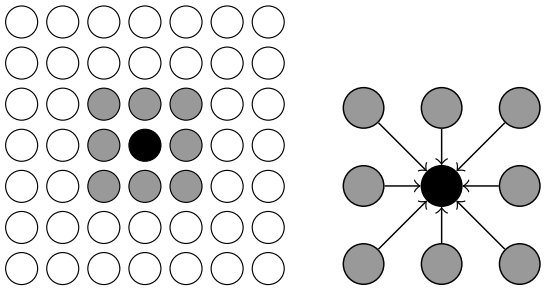
4.3.4. Göç Büyüklüğü

Bir diğer önemli parametrede göç eden bireylerin sayısıdır. Göç eden birey sayısının çok küçük olması adalar arasında paylaşılan bilginin az olması demektir ki buda algoritmaların yakınsama hızlarını yavaşlatabilmektedir. Buna karşın çok büyük olması da, göç eden bireylerin mevcut popülasyonun yerine geçmesine neden olacak ve küresel çeşitliliğin azalmasına yol açacaktır [156]. Bu sebeple göç büyüklüğünün alt popülasyon büyüklüğüne göre ayarlanmasının uygun olduğu söylenebilir.

4.4. Fine-Grained Modeli

Fine-grained modelinde popülasyondaki her bir birey bir işlemci ile temsil edilerek bireylerin uygunluk değerlerinin eş zamanlı olarak hesaplanabilmesi sağlanır [153]. Yeni nesil üretme operatörleri ise (seçim, üreme, eşleşme) küçük bir komşuluk sınırları içerisinde yakın komşuları ile yapılır ve yine yakın komşuları ile yarışılır. Bu şekilde zaman içerisinde bireylerin homojen olarak birbirleri ile etkileşimi sağlanır. Bireylerin birbirleri ile iletişim olasılığı aralarındaki mesafeye bağlıdır. Popülasyondaki her bir birey sadece yakın komşuları ile iletişim kurduğundan bu modele yerel komşuluk mekanizması da denilmektedir.

Merkezdeki bireyin iyileştirilmesi amacı ile uygulanan operatörler ve onu etkileyecek bireylerin seçimi oluşturulan komşuluk sınırları içerisinde çeşitli şekillerde yapılabilir. Örneğin komşu seçimi için, turnuva seçim yöntemi, uygunluk değerleriyle orantılı seçilme olasılığı gibi algoritmalara özgü yaklaşımlar burada kullanılır. Bunun yanında komşuluk sınırlarının oluşturulması için de farklı topolojiler kullanılabilmektedir. Genel olarak algoritmaların performansı, yakınsama hızları ve çalışma zamanları tercih edilen topolojiye bağlı olarak değişmektedir. Şekil 4.4'de modelin şematik olarak gösterimi ve Algoritma 8'de algoritmik açıklaması verilmiştir. Ayrıca Şekil 4.5'de yaygın olarak kullanılan bazı komşu belirleme

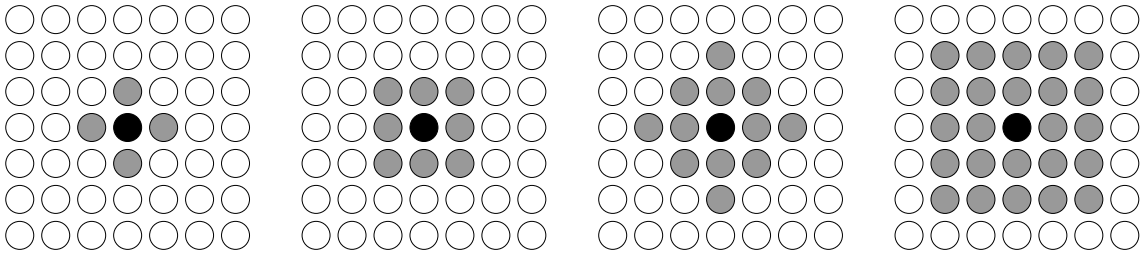


Şekil 4.4. Fine-Grained Modeli

topolojileri gösterilmiştir. Topolojilerden de görüldüğü üzere bu modelde bireyler/işlemciler arasındaki iletişim oldukça yoğundur. Bu sebeple modelin gerçekleştirimi için büyük ölçekli paralel makineler tercih edilir.

Algorithm 8: Fine-grained modeline dayalı evrimsel algoritma

```
1: for all işlemci do in parallel
2:    rastgele bir birey oluştur;
3: end for
4: while durdurma kriteri sağlanmıyorsa do
5:    for all işlemci do in parallel
6:       uygunluk değerinin hesaplanması;
7:       komşuluk sınırları içerisinde uygun komşunun seçimi;
8:       yeni birey üretme;
9:       mutasyon;
10:   end for
11: end while
```



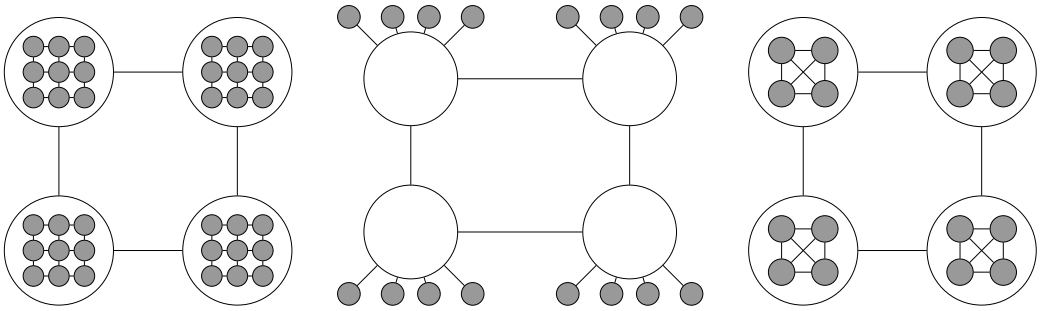
(a) (b) (c) (d)

Şekil 4.5. Fine-grained modeli için bazı komşu belirleme topolojileri, a)Linear5, b)Compact9, c)Linear13 ve d)Compact25

4.5. Hybrid Modeller

Hybrid modeller yukarıda belirtilen paralelleştirme modellerinin değişik kombinasyonları şeklinde olabilir ve genellikle iki seviyeden oluşur. Bu iki seviyeli model bir hiyerarşi olarak görülebilir. Bu tür hybrid yaklaşımların

amacı iki farklı modelin avantajlarını birleştirmektir [153]. Şekil 4.6'da bazı hybrid modellerin şematik göstermi verilmiştir. Şekil 4.6(a)'da coarse-grained ve fine-grained modelleri, Şekil 4.6(b)'de coarse-grained ve master-slave modelleri, Şekil 4.6(c)'de iki seviyeli coarse-grained modelleri ile oluşturulmuş hybrid modeller verilmiştir. Şekillerden de görüldüğü üzere genellikle iki seviyeli oluşturulan hybrid yapılarda üst seviyede coarse-grained modeli tercih edilirken alt seviyede diğer modellerin herhangi biri kullanılabilmektedir.



Şekil 4.6. Hybrid modeller, a)coarse-grained ve fine-grained, b)coarse-grained ve master-slave, c)iki seviyeli coarse-grained

4.6. Modellerin Karşılaştırılması

Görüldüğü gibi popülasyon tabanlı sezgisel algoritmalar bir çok şekilde paralelleştirilebilmektedir. Bu modeller karşılaştırıldığında ise bazı avantaj ve dezavantajlar ortaya çıkmaktadır.

Gerçek dünya problemleri gibi amaç fonksiyonlarının hesaplanmasının çok fazla zaman aldığı problemlerde tüm paralel modeller kazanç sağlamaktadır. Ancak daha kolay problemlerin çözümlerinde master-slave ve fine-grained modelleri kazanç sağlamazken coarse-grained modelinin kazanç sağladığı söylenebilir. Bunun sebebi ise problemlerin uygunluk değerlerinin hesaplanması için harcanan zamanın paralel modellerdeki işlemciler arasındaki iletişim için harcanan zamana kıyasla daha küçük ya da eşit olmasıdır. Özetle coarse-grained modelinde işlemciler arasındaki iletişim diğer modellere kıyasla çok daha azdır.

Ayrıca coarse-grained modelde popülasyonun alt popülasyonlara bölünmesi ile her bir alt popülasyonun arama uzayında farklı bölgeleri arayabilmesi ve daha çeşitli

çözümler elde edilebilmesi sağlanmaktadır. Bu sayede algoritmaların yerel optimuma takılma olasılıkları nispeten azalırken yakınsama hızları da artmaktadır. Ayrıca bu model ile algoritmalar çok farklı topolojilere kolayca uyarlanabilmektedir.

Bunun yanında coarse-grained modelinin gerektirdiği ek yeni parametreler (göç topolojisi, göç stratejileri, göç sıklığı ve göç eden birey sayısı) ise bir dezavantaj olarak görülmektedir. Çünkü bu parametreler algoritmaların performansını, yakınsama hızlarını ve çalışma zamanlarını etkilemektedir ve uygun değerlere ayarlanması gerekmektedir.

Fine-grained model işlemciler arası iletişim en fazla olduğu modeldir ve gerçekleştirimi için büyük ölçekli paralel bilgisayar sistemlerine ihtiyaç duyar. Bunun yanında coarse-grained ve master-slave modelleri paylaşımlı bellek, dağıtık bellek, kümeler, ızgaralar gibi birçok farklı donanımlar ile oluşturulabilen paralel bilgisayar sistemlerinde gerçekleştirilebilir.

Ayrıca farklı modellerin avantajlarının birleştirildiği hybrid modeller çok farklı kombinasyonlar da olabilmektedir. Ancak bu kombinasyonlar ile algoritmalar daha verimli hale getirilseler de, ortaya çıkan yeni parametreler sistemleri karmaşıklaştırarak bir dezavantaj oluşturmaktadır.

Modellerin getirdiği bazı avantaj ve dezavantajların yanısıra gerçekleştirimi yapılacak sezgisel algoritmanın paralel modellerle etkileşimi son derece öndemlidir. Kısaca master-slave modelinin senkron veya asenkron gerçekleştirimi, fine-grained modelinde belirlenen komşuluk sınırları, coarse-grained modelinde belirlenen göç topolojileri, göç büyüklüğü, göç sıklığı gibi parametreler algoritmaların performanslarını etkileyebilmekte ve bu etkiler algoritmaların karakteristiklerine göre olumlu yada olumsuz yönde olabilmektedir. Bu sebeple en iyi paralelleştirme modelinin seçimi ve uygun parametre değerlerinin ayarlanması sezgisel algoritmanın kendisine bağlıdır denilebilir. Ancak çözülmek istenen problemin uygunluk değerinin hesaplanma süresi, örneklerin boyutu gibi bazı özellikler önceden bilinirse, uygun modelin seçilmesi kolaylaşabilir.

5. BÖLÜM

BULGULAR

5.1. Giriş

Gelişen teknolojiler, gerçek dünya problemlerinin daha da büyümesine ve zorlaşmasına sebep olmaktadır. Bu tür büyük boyutlu ve zor problemlerin çözümünde artan hesaplama ihtiyaçlarını karşılamak için de araştırmacılar paralel hesaplama sistemlerine yönelmişlerdir. Paralel hesaplama ile problemler parçalara bölünmekte, bu parçaların eş zamanlı olarak çözülmesi ile de sonuçların daha hızlı elde edilmesi amaçlanmaktadır.

Tez kapsamında sezgisel algoritmaların paralelleştirilmesinde kullanılan temel modeller gerçekleştirilerek paralel hesaplama sistemlerinin kullanıldığı farklı uygulamalar yapılmıştır. Ayrıca gerçekleştirilen sezgisel algoritmalardan bazıları için performanslarını arttırıcı yeni yaklaşımlar önerilmiştir. Önerilen yaklaşımlar ve paralel gerçekleştirimlerin performans ve çalışma zamanları, değişik özelliklere sahip zorluk derecesi yüksek problemler üzerinde test edilmiştir. Gerçekleştirilen her bir uygulama bölüm içerisinde farklı bir alt başlık altında verilmiştir.

İlk uygulamada temelde asenkron çalışan ABC algoritmasının senkron modeli önerilmiştir. Senkron ABC algoritmasının master-slave parallelleştirme modeli gerçekleştirilerek analiz edilmiştir. Bir diğer uygulamada PSO algoritmasının performansını geliştirmek için yeni bir yaklaşım önerilmiştir. Bu yaklaşım ile PSO algoritmasının çözüm kalitesi iyileştirilirken, fine-grained parallelleştirme modelinin analizleri geliştirilmiş bu model üzerinde yapılmıştır. Başka bir uygulamada ABC algoritmasının coarse-grained parallelleştirme modeli gerçekleştirilerek, bu

modele özgü kontrol parametrelerinden alt popülasyon sayısı, göç topolojisi ve göç sıklığı değerlerinin performans üzerindeki etkisi incelenmiştir. Bir başka uygulamada, literatürde çok yeni bir sezgisel algoritma olan TLBO algoritmasının master-slave, coarse-grained ve hybrid paralelleştirme modelleri incelenmiştir. Başka bir uygulamada literatürdeki güncel sezgisel algoritmalardan ABC, CS, DE, FF, GS, PSO ve TLBO algoritmalarının seri ve coarse-grained paralel modelleri performans ve çalışma zamanı bakımından karşılaştırılmıştır. Bir diğer uygulama da DE algoritmasının stratejiye olan performans bağımlılığını önlemek için yeni bir yaklaşım önerilmiştir. Önerilen yaklaşımın paralel gerçekleştirimi ile de hem performans hem zamandan kazanç sağlanmıştır. Başka bir uygulamada paralel ABC algoritmasının yapay sinir ağları eğitiminde kullanılması ile bazı sınıflandırma problemlerinin çözümü gerçekleştirilmiştir. Son uygulamamızda ise paralel ABC algoritmasının ayrık modeli geliştirilmiş ve etkin komşu üretme mekanizmaları entegre edilmiştir. Geliştirilen model ile ayrık bir problem türü olan gezgin satıcı problemlerinin bazılarının çözümü gerçekleştirilmiştir.

Uygulamaların tamamı C programlama dilinde MPI paralel programlama kütüphanesi kullanılarak kodlanmıştır [137]. Uygulama çözümleri TÜBİTAK ULAKBİM, Yüksek Başarım ve Grid Hesaplama Merkezindeki bilgisayar sistemlerinde gerçekleştirilmiştir. Hesaplama merkezindeki bilgisayar sistemlerinin teknik özellikleri Tablo 5.1'de verilmiştir.

Uygulamalarda gerçekleştirilen test fonksiyonları çözümlerinde CEC2008 [159], CEC2010 [160] ve bazı yaygın kullanılan büyük ölçekli zorlaştırılmış sürekli test fonksiyon dikkate alınmıştır. Tablo 5.2'de verilen bu fonksiyonlardan her biri farklı zorluk derecesine ve karakteristiğe sahiptir. Tablodaki D problem boyutunu gösterirken, m parametresi her gruptaki değişkenlerin sayısını kontrol etmek için kullanılan ayrılabilirlik derecesidir. m parametresi CEC2010 fonksiyonları için 50 olarak alınmıştır. Fonksiyonlarla ilgili detaylı bilgiler EK-1'de bulunmaktadır. İstatistiksel testlerin güvenilirliği açısından problemlerin tamamı 30'ar kez çözülmüştür. Sonuç tablolarında bu 30 koşmadan elde edilen ortalama ve standart sapma değerleri verilmiştir.

Tablo 5.1. Deneysel çalışmalarda kullanılan paralel hesaplama sistemi özellikleri

İşlemci	2x AMD Opteron 6176 (24 çekirdek)
Bellek	128 GB 1600 MHz Ecc
Yerel Disk	1TB (Raid 0 2x500 GB)
Bağlantı	40Gbps QDR infiniband
Kaynak yöneticisi	SLURM

Tablo 5.2. Deneysel çalışmalarda kullanılan nümerik test fonksiyonları(D:Problem boyutu, m:ayrılabilirlik derecesi)

	Fonksiyon	Aralık	Kaynak
F_1	Shifted Sphere	$[-100, 100]^D$	CEC2008
F_2	Shifted Schwefel 2.21	$[-100, 100]^D$	CEC2008
F_3	Shifted Rosenbrock	$[-100, 100]^D$	CEC2008
F_4	Shifted Rastrigin	$[-5, 5]^D$	CEC2008
F_5	Shifted Griewank	$[-600, 600]^D$	CEC2008
F_6	Shifted Ackley	$[-32, 32]^D$	CEC2008
F_7	Step	$[-100, 100]^D$	-
F_8	Quartic	$[-1.28, 1.28]^D$	-
F_9	Schwefel 2.22	$[-10, 10]^D$	-
F_{10}	Dixon Price	$[-10, 10]^D$	-
F_{11}	Penalized	$[-50, 50]^D$	-
F_{12}	Penalized2	$[-50, 50]^D$	-
F_{13}	Shifted Elliptic	$[-100, 100]^D$	CEC2010
F_{14}	Shifted Rastrigin	$[-5, 5]^D$	CEC2010
F_{15}	Shifted Ackley	$[-32, 32]^D$	CEC2010
F_{16}	Single-group shifted and m-rotated Elliptic	$[-100, 100]^D$	CEC2010
F_{17}	Single-group shifted and m-rotated Rastrigin	$[-5, 5]^D$	CEC2010
F_{18}	Single-group shifted and m-rotated Ackley	$[-32, 32]^D$	CEC2010
F_{19}	Single-group shifted m-dimensional Schwefel	$[-100, 100]^D$	CEC2010
F_{20}	Single-group shifted m-dimensional Rosenbrock	$[-100, 100]^D$	CEC2010
F_{21}	$\frac{D}{2m}$-group shifted and m-rotated Elliptic Fonksiyon	$[-100, 100]^D$	CEC2010
F_{22}	$\frac{D}{2m}$-group shifted and m-rotated Rastrigin	$[-5, 5]^D$	CEC2010
F_{23}	$\frac{D}{2m}$-group shifted and m-rotated Ackley	$[-32, 32]^D$	CEC2010
F_{24}	$\frac{D}{2m}$-group shifted m-dimensional Schwefel	$[-100, 100]^D$	CEC2010
F_{25}	$\frac{D}{m}$-group shifted m-dimensional Rosenbrock	$[-100, 100]^D$	CEC2010
F_{26}	$\frac{D}{m}$-group shifted and m-rotated Elliptic Fonksiyon	$[-100, 100]^D$	CEC2010
F_{27}	$\frac{D}{m}$-group shifted and m-rotated Rastrigin	$[-5, 5]^D$	CEC2010
F_{28}	$\frac{D}{m}$-group shifted and m-rotated Ackley	$[-32, 32]^D$	CEC2010
F_{29}	$\frac{D}{m}$-group shifted m-dimensional Schwefel	$[-100, 100]^D$	CEC2010
F_{30}	$\frac{D}{m}$-group shifted m-dimensional Rosenbrock	$[-100, 100]^D$	CEC2010
F_{31}	Shifted Schwefel's problem 1.2	$[-100, 100]^D$	CEC2010
F_{32}	Shifted Rosenbrock	$[-100, 100]^D$	CEC2010

5.2. Senkron ABC Algoritmasının Master-Slave Paralel Gerçekleştirimi

Bu uygulamada ABC algoritmasının master-slave paralelleştirme modelinin incelenmesi amaçlanmıştır. Master-slave paralelleştirme modeli senkron ve asenkron olmak üzere iki farklı şekilde gerçekleştirilebilmektedir.

Orjinal ABC algoritması asenkron olarak çalışmaktadır. Asenkron algoritmalarda; her bir yeni çözüm oluşturulur, değerlendirilir ve hemen karşılaştırılarak bir sonraki adıma geçilir. Sonraki bireylerde popülasyonun değişmiş hali üzerinden komşu seçimi gerçekleşir. Diğer taraftan senkron algoritmalarda ise yeni çözümlerin tamamı oluşturulur, tamamı değerlendirilir ve karşılaştırılır. Tüm bireyler aynı popülasyonu kullanır. Algoritma 9'da sözde kodu verilen asenkron ABC algoritmasında açgözlü seleksiyon ve güncelleme işlemi yeni bir kaynak üretilip, uygunluk değeri hesaplandıktan hemen sonra gerçekleştirilmektedir. Buna karşın Algoritma 10'da sözde kodu verilen senkron ABC algoritması ise kaynakların uygunluk değerlerinin hesaplanması, açgözlü seleksiyon ve güncelleme işlemi tüm yeni kaynaklar üretildikten sonra gerçekleştirilmektedir.

Narasimhan ABC algoritmasında tüm koloniyi işlemciler arasında paylaştırmıştır [79]. Elde edilen çözümler hem yerel bellekte hemde paylaşımlı bellekte tutulmaktadır. Her iterasyonda işlemcideki arılar yerel bellekteki çözümleri iyileştirmeye çalışmaktadır. İterasyon sonunda ise çözümler paylaşımlı belleğe kopyalanarak diğer arıların kullanımına sunulmaktadır. Yapılan bu çalışmada çözüm kalitesini düşürmeden hızlanma sağlanmıştır.

Banharnsakun ve arkadaşları ABC algoritmasının çalışma süresini ve kaynak kullanımını indirgemek için algoritmanın dağıtık versiyonunu geliştirmişlerdir [161]. Geliştiridikleri modelde populasyon alt gruplara bölünmüş ve her biri bir işlemcide çalışarak lokal arama yapmaları şağlanmıştır. İşlemcilerin yönetiminde yönetici-işçi (manager-worker) model kullanmışlardır [128]. Yönetici işlemci; algoritmanın başlatılmasından, yük dengesinden, popülasyonun bölümlenmesinden, alt gruplarının işçilere dağıtılmasından, global sonuçların toplanmasından ve görüntülenmesinden sorumludur. İşçiler ise asıl hesaplamaları yapmaktadır. Yöneticiden aldıkları verileri işleyerek o anki lokal en iyi çözümlerini geri gönderirler.

Her iterasyonun sonunda yönetici tüm popülasyondan iki rastgele alt grubu seçer ve bu alt gruplar aralarında lokal en iyi çözümlerini paylaşırlar. Bir gruptaki lokal en iyi çözüm diğerindeki en kötü çözümle değiştirilir. Sadece bu noktada haberleştikleri için haberleşme maliyeti nispeten düşüktür. Son iterasyonda yönetici tüm işçilerden lokal en iyi çözümlerini alarak küresel en iyi çözümü hesaplar. Yapılan bu çalışmada haberleşme yapısı MPI üzerinden sağlanmıştır.

Parpinelli ve arkadaşları master-slave, alt populasyonlar arası iletişime izin veren çoklu alt populasyon ve hybrid yapı olmak üzere üç farklı parallelştirme modelini incelemişlerdir [108]. Master-slave yapıda tek bir popülasyon bulunmaktadır. Master işlemci popülasyonun başlatılmasından, algoritmanın adımlarından ve arıların slave işlemcilere dağıtılmasından sorumludur. Slave işlemciler amaç fonksiyonu hesaplamalarından ve sonuçların master işlemciye geri gönderilmesinden sorumludur. Alt popülasyon modelinde ise birden fazla popülasyon bulunmaktadır ve coarse-grained yapı üzerine inşa edilmiştir. Farklı başlatma şartları ile iki yada daha fazla popülasyon başlatılır ve her bir popülasyon bir ada olarak düşünülür. Her biri bir işlemcide olmak üzere çalıştırılır ve belli peryotlarda bu alt popülasyonlar haberleştirilir. Paralel hibrid modelde üst seviyede çoklu popülasyonlu coarse-grained adalar bulunmakta alt seviye de ise tek popülasyonlu master-slave modeli bulunmaktadır.

Luo ve arkadaşları yapay arıların alt popülasyonlara bölündüüğü ve her alt popülasyonda bağımsız ABC algoritmasının çalıştığı bir paralel yapay arı kolonisi algoritması önermişlerdir [98]. Her alt popülasyonun kendi bireyleri ve en iyi çözümü bulunmaktadır. Belli peryotlarda alt popülasyonların en kötü k bireyi kendilerinden önce gelen alt popülasyonun en iyi k bireyi ile değiştirilerek haberleşmeleri sağlanmıştır. Böylece farklı popülasyonlardaki bireyler arasında bilgi akışı sağlanır.

Subotic ve arkadaşları farklı popülasyonlara farklı işler atayarak popülasyonlar arasında haberleşmenin gerçekleştiği bir paralel ABC önermişlerdir [97]. Haberleşme sıklığı keşif (exploration) ve bilgiden faydalanma (exploitation) süreçleri incelenmiştir.

Benitez ve Lopes ABC algoritması için master-slave ve hibrid yapı olmak üzere

iki paralel yaklaşım önermişlerdir [107]. Dört farklı deney örneği üzerinde gerçekleştirilen modelleri algoritmanın seri versiyonu ile kıyaslamışlardır.

Hong ve arkadaşları MPI ile ABC algoritmasını paralelleştirmiş ve kompleks optimizasyon problemleri üzerinde seri ve paralel versiyonlarını kıyaslamışlardır [162].

Yapılan bu çalışmalarda az sayıda işlemci ile ABC algoritmasının asenkron modeli incelenmiştir. İşlemci sayısının artmasının paralel modellerin hızlanma ve verimlilik üzerindeki etkileri incelenmemiştir. Bir başka konuda yapılan bu çalışmalarda yüksek hesaplama gerektiren problemlerin dikkate alınmamış olmasıdır.

Senkron yapay arı koloni algoritmasının (Synchronous Artificial Bee Colony, SABC) gerçekleştirildiği ve senkron ve asenkron modellerinin karşılaştırıldığı bir çalışma 2012 yılında bu tez kapsamında gerçekleştirilmiştir [80].

Önerilen senkron master-slave paralel yapay arı kolonisi algoritmasında (Parallel Synchronous Artificial Bee Colony, PSABC) kaynakların uygunluk değerleri slave işlemciler tarafından eş zamanlı olarak hesaplanmaktadır. Master işlemcide gerçekleştirilen seleksiyon, güncelleme ve olasılık değerlerinin hesaplanması gibi işlemler tüm yeni kaynakların uygunluk değerleri hesaplanıp slave işlemcilerden teslim alındıktan sonra gerçekleşmektedir. Modelin daha iyi anlaşılabilmesi için master ve slave işlemcilerin görevleri maddeler halinde verilmiştir.

Yapılan bu uygulamada, önerilen modelin performans ve çalışma zamanı CEC2010 için özel olarak hazırlanmış büyük ölçekli ve zorlaştırılmış yirmi adet sürekli test fonksiyonu [160] üzerinde incelenmiştir. Tüm test fonksiyonlarının çözümü CEC2010 için tanımlanan problem boyutunda ve iterasyon sayısında gerçekleştirilmiştir. Tanımlanan bu değerler problem boyutu için 1000 ve iterasyon sayısı için 3000000 dur. Sonuçların karşılaştırılması için popülasyon büyüklüğü 100, algoritmaya özel "limit" parametre değeri 100 ve 2000 olmak üzere iki farklı değer alınmıştır. Önerilen modelin paralel performansını ölçmek için literatürde yaygın olarak kullanılan yöntemlerden hızlanma ve verimlilik ölçümleri dikkate alınmıştır. Verimlilik ve paralel hızlanma değerleri için 4, 8, 16, 32 ve 48 adet işlemci kullanılmıştır.

Algorithm 9: Seri asenkron ABC algoritması için sözde kod

```
algoritma parametrelerini ayarla;
başlangıç popülasyonunun oluştur;
for iter ← 1 to MaxCycle do
  for i ← 1 to Foods do
    işçi arı için yeni bir kaynak üret;
    kaynağın uygunluk değerini hesapla;
    açgözlü seleksiyon işlemi uygula ve daha iyi olanı seç;
  end for
  uygunluk değerine dayalı olasılık değerlerini hesapla;
  t = 0;
  while t < Foods do
    if rand() < p_i then
      gözcü arı için yeni bir kaynak üret ;
      kaynağın uygunluk değerini hesapla;
      aç gözlü seleksiyon işlemi uygula ve daha iyi olanı seç;
      t = t + 1;
    end if
    i = i + 1;
    if i = Foods then
      i = 0;
    end if
  end while
  kâşif arı safhasını çalıştır;
  en iyi çözümü hafızada tut;
end for
```

Algorithm 10: Seri SABC algoritması için sözde kod

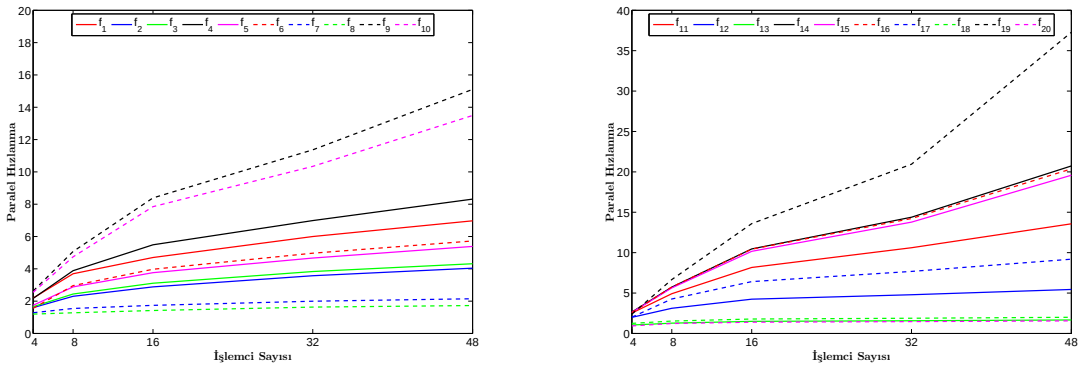
```
algoritma parametrelerini ayarla;
başlangıç popülasyonunun oluştur;
for iter ← 1 to MaxCycle do
  for i ← 1 to Foods do
    işçi arı için yeni bir kaynak üret;
  end for
  kaynakların uygunluk değerlerini hesapla;
  aç gözlü seleksiyon işlemi uygula ve daha iyi olanları seç;
  uygunluk değerine dayalı olasılık değerlerini hesapla;
  t = 0;
  while t < Foods do
    if rand() < p_i then
      gözcü arı için yeni bir kaynak üret ;
      t = t + 1;
    end if
    i = i + 1;
    if i = Foods then
      i = 0;
    end if
  end while
  kaynakların uygunluk değerlerini hesapla;
  aç gözlü seleksiyon işlemi uygula ve daha iyi olanları seç;
  kâşif arı safhasını çalıştır;
  en iyi çözümü hafızada tut;
end for
```

- **Master işlemci**

  1. Algoritma parametrelerinin ayarlanması, başlangıç popülasyonunun oluşturulması
  2. Yeni kaynaklar üretilmesi
  3. Üretilen yeni kaynakların slave işlemcilere gönderilmesi
  4. Slave işlemcilerde hesaplanan uygunluk değerlerinin teslim alınması
  5. Aç gözlü seleksiyon metodu ile daha iyi olan çözümlerin seçilmesi
  6. Uygunluk değerine dayalı olasılık değerlerininin hesaplaması
  7. Kâşif arı biriminin çalıştırılması
  8. En iyi çözümün hafızada tutulması

- **Slave işlemci**

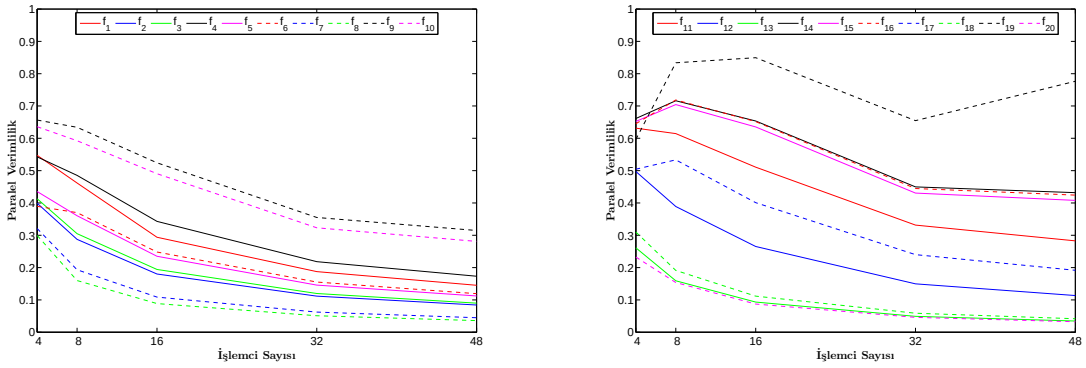
  1. Master işlemciden üretilen yeni kaynağın teslim alınması
  2. Kaynağın uygunluk değerinin hesaplanması
  3. Master işlemciye hesaplanan uygunluk değerinin gönderilmesi



(a) (b)

Şekil 5.1. PSABC algoritması için paralel hızlanma grafikleri a) f_1-f_{10}, b) f_{11}-f_{20}

Tablo 5.3'de ABC ve SABC algoritmaları ile test fonksiyonlarının çözümleri sonucu elde edilen ortalama uygunluk ve standart sapma değerleri farklı "limit" parametre değerleri için karşılaştırmalı verilmiştir. Tablo 5.4'de PSABC algoritması ile farklı işlemci sayıları ile gerçekleştirilen test fonksiyonlarının çözüm süreleri ve elde



(a) (b)

Şekil 5.2. PSABC algoritması için paralel verimlilik grafikleri a) f_1-f_{10}, b) f_{11}-f_{20}

edilen paralel hızlanma değerleri verilmiştir. Ayrıca Şekil 5.1 ve Şekil 5.2'de test fonksiyonlarının sırası ile hızlanma ve verimlilik grafikleri verilmiştir.

Gerçekleştirilen bu uygulamada iki farklı inceleme yapılmıştır. a) seri model üzerinde asenkron ABC ve önerilen SABC algoritmalarının çözüm kaliteleri açısından karşılaştırılması, b) SABC algoritmasının master-slave paralel gerçekleştirimi yapılarak paralel hızlanma ve verimlilik ölçümlerinin incelenmesi.

Tablo 5.3'deki sonuçlar incelendiğinde SABC algoritması en az asenkron ABC algoritması kadar etkili çözümler üretebilmektedir. Ayrıca "limit" değerinin artması popülasyondaki çeşitliliğin yeterli olması sebebi ile asenkron ABC ve SABC algoritmalarının performanslarını önemli ölçüde değiştirmemiştir.

Tablo 5.4'deki sonuçlar incelendiğinde f_{14}, f_{15}, f_{16} ve f_{19} fonksiyonlarının uygunluk değerlerinin hesaplanma sürelerinin çok uzun olduğu görülmektedir. Bu sebeple bu fonksiyonlarda PSABC algoritması ile elde edilen hızlanma değerleri diğer fonksiyonlara göre çok yüksek çıkmıştır. Özellikle uygunluk değerinin hesaplanma süresinin en uzun olduğu f_{19} fonksiyonunda önemli bir hızlanma elde edilmiştir. Diğer taraftan f_8, f_{13} ve f_{20} fonksiyonlarının uygunluk değerlerinin hesaplanma süreleri çok kısa olduğu için önemli bir hızlanma elde edilememiştir.

Tablo 5.3. Farklı "limit" değerleri ile koşulan ABC ve SABC algoritmalarının ortalama ve standart sapma değerleri, Ort.: En iyi değerlerin ortalamaları, Std.: En iyi değerlerin standart sapmaları

	Fonksiyon	f_1		f_2		f_6		f_7		f_{12}	
Limit	Algoritma	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.
100	ABC	5.21E-05	7.13E-05	7.22E+01	1.34E+01	2.01E+07	9.17E+04	5.72E+10	6.21E+09	7.45E+05	4.56E+04
	SABC	1.83E-05	3.05E-05	7.81E+01	1.23E+01	2.01E+07	7.64E+04	5.37E+10	8.50E+09	7.40E+05	3.44E+04
2000	ABC	1.46E-06	1.35E-06	7.08E+01	1.16E+01	2.00E+07	6.95E+04	4.51E+10	5.80E+09	7.42E+05	3.05E+04
	SABC	1.87E-06	9.55E-07	7.53E+01	1.19E+01	2.01E+07	5.96E+04	4.31E+10	7.73E+09	7.42E+05	3.81E+04
	Fonksiyon	f_{13}		f_{14}		f_{15}		f_{19}		f_{20}	
Limit	Algoritma	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.
100	ABC	6.60E+02	1.09E+02	2.01E+09	1.20E+08	1.57E+04	2.84E+02	1.09E+07	6.02E+05	3.54E+02	1.34E+02
	SABC	6.45E+02	1.47E+02	2.00E+09	1.01E+08	1.56E+04	2.42E+02	1.09E+07	5.88E+05	4.06E+02	1.71E+02
2000	ABC	6.68E+02	1.89E+02	1.79E+09	7.79E+07	1.46E+04	2.61E+02	8.66E+06	4.00E+05	5.17E+02	1.76E+02
	SABC	6.94E+02	1.75E+02	1.81E+09	9.01E+07	1.46E+04	1.93E+02	8.53E+06	3.98E+05	6.20E+02	1.41E+02

Tablo 5.4. SABC ve farklı CPU sayıları ile koşulan PSABC algoritmalarının çalışma süreleri ve paralel hızlanma değerleri, SU: Paralel hızlanma

	SABC	PSABC									
CPU Sayısı	1	4		8		16		32		48	
Fonksiyon	Zaman	Zaman	SU	Zaman	SU	Zaman	SU	Zaman	SU	Zaman	SU
f_1	495.1012	226.1350	2.19	134.0360	3.69	105.3560	4.70	82.5440	6.00	71.0180	6.97
f_2	260.3244	163.0100	1.60	113.2340	2.30	90.3920	2.88	72.8740	3.57	64.4600	4.04
f_3	268.7792	162.7460	1.65	110.2120	2.44	86.4920	3.11	70.1840	3.83	62.2940	4.31
f_4	583.7040	268.9320	2.17	150.3480	3.88	106.4040	5.49	83.5680	6.98	70.2040	8.31
f_5	355.5488	204.2100	1.74	123.3360	2.88	94.5600	3.76	76.1980	4.67	66.0260	5.38
f_6	366.8332	235.4600	1.56	123.8980	2.96	92.3700	3.97	73.8740	4.97	64.0940	5.72
f_7	130.5988	101.7380	1.28	84.5120	1.55	75.1580	1.74	65.4920	1.99	60.8360	2.15
f_8	105.7196	88.2320	1.20	82.7140	1.28	74.5320	1.42	64.8500	1.63	61.2380	1.73
f_9	1288.6042	490.6800	2.63	254.0800	5.07	153.6420	8.39	113.4080	11.36	85.3140	15.10
f_{10}	1091.0761	428.9600	2.54	230.2200	4.74	139.0520	7.85	105.5380	10.34	80.8840	13.49
f_{11}	1094.3417	433.2520	2.53	222.5680	4.92	133.9400	8.17	103.1880	10.61	80.6660	13.57
f_{12}	358.7124	180.3080	1.99	115.3040	3.11	84.5620	4.24	74.9000	4.79	65.8620	5.45
f_{13}	103.9984	99.5940	1.04	81.7260	1.27	69.6560	1.49	66.2360	1.57	62.5300	1.66
f_{14}	2078.7385	785.8120	2.65	362.8160	5.73	198.7920	10.46	144.4940	14.39	100.2720	20.73
f_{15}	1904.1840	729.7980	2.61	337.9080	5.64	187.3460	10.16	138.2120	13.78	97.2640	19.58
f_{16}	1931.7239	748.6300	2.58	336.0000	5.75	185.2560	10.43	135.7940	14.23	94.8600	20.36
f_{17}	641.3616	317.9240	2.02	150.3380	4.27	99.9940	6.41	83.5460	7.68	69.7660	9.19
f_{18}	124.3004	101.8320	1.22	81.2720	1.53	69.6420	1.78	66.1360	1.88	62.0540	2.00
f_{19}	9820.4580	4115.7135	2.39	1472.5399	6.67	722.5599	13.59	468.7820	20.95	263.4540	37.28
f_{20}	96.5776	103.7960	0.93	78.5700	1.23	69.3220	1.39	65.8320	1.47	62.0300	1.56

Ayrıca, işlemci sayısının etkisinin gösterilmesi amacı ile işlemci sayısı 48'e kadar arttırılmıştır. İşlemci sayısının artması işlemciler arası iletişim giderlerini arttırmış, çok sayıda slave işlemcinin aynı anda master işlemci ile iletişim kurmak istemesi sebebi ile de iletişimde darboğazlara sebep olmuştur. Sonuçlardan artan işlemci sayısı ile paralel hızlanmanın önemli ölçüde artmadığı ve verimlilik değerinin düştüğü gözlemlenmiştir.

Özetle önerilen PSABC algoritması asenkron ABC algoritması ile performans açısından çok farklılık göstermezken fonksiyon hesaplamalarının masraflı ve zaman alıcı olduğu problemlerde çalışma zamanı bakımından avantajlı olacağı söylenebilir.

5.3. Geliştirilmiş PSO Algoritmasının Fine-Grained Paralel Gerçekleştirimi

Bu uygulamada PSO algoritmasının performansının iyileştirilmesi amacı ile yeni bir model önerilmiş ve önerilen bu model üzerinde fine-grained paralelleştirme modelinin incelenmesi amaçlanmıştır.

Bölüm 3'de detaylı olarak anlatılan temel PSO algoritmasında yeni üretilen aday çözümler herhangi bir karşılaştırma yapmadan mevcut çözümlerin yerini almaktadır. Genel olarak algoritma da üretilen yeni çözümlerin mevcut çözümün kalitesini koruyacağı ya da iyileştireceği düşünülmektedir. Fakat uygulamalarda zaman zaman yeni üretilen iyileştirilmiş aday çözümler arama uzayında çok farklı bölgelere dağılabilmekte ve iyileşmenin aksine daha da kötüleşebilmektedir. PSO algoritmasındaki bu dezavantajı gidermek amacı ile önerilen geliştirilmiş modelde her bir iterasyonda n adet kaliteli aday çözümün alınarak hiç değiştirmeden bir sonraki iterasyona taşınması amaçlanmıştır. Bir sonraki iterasyona taşınan kaliteli aday çözümler mevcut popülasyondaki en kötü n adet bireyin yerini almaktadır. Bu sayede en iyi bireyler muhafaza edilerek ortalama kalite yükseltilmekte ve en iyi bireylerin bulunduğu bölgelerin daha fazla araştırılması sağlanmaktadır. Bu yapıda ayrıca çeşitliliği sağlamak amacıyla aynı amaç fonksiyon değerini üreten aday çözümler tespit edilerek bir tanesi muhafaza edilirken bir diğeri popülasyondan çıkarılarak yerine rasgele üretilen yeni bir aday çözüm yerleştirilmektedir. Önerilen geliştirilmiş PSO algoritmasının (Improved Particle Swarm Optimization, IPSO)

temel adımları Algoritma 11'de verilmiştir.

Algorithm 11: IPSO algoritmasının temel adımları

```
1: Başlangıç popülasyonunun oluşturulması
2: Uygunluk değerinin hesaplanması
3: En iyi çözümün belirlenmesi;
4: repeat
5:    En iyi n adet çözümün hafızaya alınması;
6:    Hız vektörünün oluşturulması;
7:    Yeni aday çözüm üretilmesi;
8:    Uygunluk değerinin hesaplanması
9:    Güncelleme
10:   En kötü n adet çözümün yerine hafızadaki n adet iyi çözümün konulması;
11:   Çift aday çözümlerin yerine rastgele yeni çözümler üretilmesi;
12:   En iyi çözümün belirlenmesi;
13:   Kendi en iyi değerinin güncellenmesi;
14: until
```

IPSO algoritması temel PSO algoritmasının kendine özgü parametrelerine ek olarak, bir sonraki iterasyona aktarılacak en iyi birey sayısının da tanımlanmasını gerektirir. Yapılan çalışma da, CEC2008 için özel olarak hazırlanmış zorlaştırılmış altı adet test fonksiyonu ile yaygın olarak kullanılan altı adet test fonksiyonu kullanılmıştır. Test fonksiyonları için problem boyutları ve çözüm için kullanılacak iterasyon sayıları sırası ile 100 ve 500000 alınmıştır. Ayrıca algoritmaya özgü parametrelerden $w = 0.6$, öğrenme faktörleri $c_1 = 1.8$ ve $c_2 = 1.8$ alınmıştır. Bir sonraki iterasyona aktarılacak en iyi birey sayısı ise 2, 4, 8 ve 16 olacak şekilde dört farklı değer için incelenmiştir. Bu amaçla Tablo 5.5'de PSO ve IPSO algoritmaları ile test fonksiyonlarının çözümleri sonucu elde edilen ortalama uygunluk ve standart sapma değerleri farklı en iyi birey sayıları için karşılaştırmalı verilmiştir.

Ayrıca paralelleştirme modellerinden fine-grained modeli IPSO algoritması ile gerçekleştirilerek paralel IPSO algoritması (Parallel Improved Particle Swarm Optimization, PIPSO) oluşturulmuştur. Fine-grained paralelleştirme modeli farklı komşuluk topolojileri ile gerçekleştirilebilmektedir. Bu topolojiler algoritmaların performanslarını, yakınsama hızlarını ve çalışma zamanlarını etkilemektedir. Yaygın olarak kullanılan Linear5, Compact9, Linear13 ve Compact25 topolojileri bu çalışmada kullanılmıştır. Tablo 5.4'de IPSO ve farklı komşu belirleme topolojileri ile koşulan PIPSO algoritmalarının ortalama ve standart sapma değerleri verilmiştir.

Tablo 5.5. PSO ve farklı en iyi birey sayıları ile koşulan IPSO algoritmalarının ortalama ve standart sapma değerleri, Ort.: En iyi değerlerin ortalamaları, Std.: En iyi değerlerin standart sapmaları

Fonksiyon	PSO		IPSO							
			2		4		8		16	
	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.
f_1	1.77E+04	4.59E+03	2.90E+01	1.15E+01	3.19E+01	1.22E+01	1.90E+01	8.78E+00	1.72E+01	1.42E+01
f_2	5.68E+01	1.25E+01	3.48E+01	6.66E+00	3.06E+01	5.99E+00	3.23E+01	4.38E+00	3.04E+01	4.31E+00
f_3	2.19E+09	2.01E+09	2.51E+04	1.11E+04	3.50E+04	2.09E+04	2.62E+04	1.44E+04	1.74E+04	1.32E+04
f_4	1.23E+03	1.88E+02	2.12E+01	5.13E+00	1.92E+01	4.94E+00	1.90E+01	3.72E+00	1.87E+01	4.36E+00
f_5	1.72E+02	5.88E+01	1.26E+00	1.04E-01	1.30E+00	1.27E-01	1.20E+00	1.14E-01	1.12E+00	8.43E-02
f_6	1.30E+01	1.29E+00	2.22E+00	1.70E-01	2.23E+00	2.73E-01	2.13E+00	1.99E-01	2.12E+00	1.88E-01
f_7	4.85E+03	4.74E+03	4.85E+02	2.58E+02	5.03E+02	2.87E+02	4.37E+02	2.35E+02	3.71E+02	3.17E+02
f_8	6.80E+00	1.72E+00	7.72E-01	1.70E-01	7.30E-01	1.97E-01	6.87E-01	1.85E-01	6.81E-01	1.83E-01
f_9	5.26E+01	3.93E+01	1.99E+00	4.22E-01	1.82E+00	4.09E-01	1.25E+00	3.84E-01	7.84E-01	2.27E-01
f_{10}	4.92E+04	3.95E+04	9.25E+01	2.50E+01	9.40E+01	2.58E+01	8.34E+01	2.24E+01	5.73E+01	1.43E+01
f_{11}	1.61E+05	2.59E+05	8.08E-02	6.37E-02	5.64E-02	5.13E-02	3.26E-02	3.63E-02	2.36E-02	3.50E-02
f_{12}	2.61E+06	3.00E+06	3.62E+00	1.27E+00	3.28E+00	1.31E+00	2.21E+00	9.74E-01	1.74E+00	2.45E-02

Tablo 5.6. IPSO ve farklı komşu belirleme topolojileri ile koşulan PIPSO algoritmalarının ortalama ve standart sapma değerleri, Ort.: En iyi değerlerin ortalamaları, Std.: En iyi değerlerin standart sapmaları

Fonksiyon	IPSO		PIPSO							
			*Linear*5		*Compact*9		*Linear*13		*Compact*25	
	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.
f_1	1.72E+01	1.42E+01	7.92E+01	6.66E+01	4.30E+01	1.97E+01	4.16E+01	4.05E+01	1.83E+01	1.26E+01
f_2	3.04E+01	4.31E+00	4.54E+01	5.69E+00	4.73E+01	5.34E+00	4.74E+01	7.48E+00	4.57E+01	6.31E+00
f_3	1.74E+04	1.32E+04	2.55E+05	1.76E+05	1.38E+05	7.81E+04	1.08E+05	1.41E+05	3.73E+04	4.27E+04
f_4	1.87E+01	4.36E+00	2.44E+01	4.45E+00	2.05E+01	3.20E+00	1.93E+01	4.70E+00	1.84E+01	2.52E+00
f_5	1.12E+00	8.43E-02	1.60E+00	2.11E-01	1.29E+00	1.33E-01	1.25E+00	1.69E-01	1.18E+00	1.13E-01
f_6	2.12E+00	1.88E-01	2.34E+00	2.44E-01	2.25E+00	2.30E-01	2.26E+00	2.12E-01	2.19E+00	2.00E-01
f_7	3.71E+02	3.17E+02	2.88E+02	1.84E+02	8.07E+02	9.89E+02	1.65E+03	1.98E+03	7.79E+02	6.73E+02
f_8	6.81E-01	1.83E-01	3.11E+00	6.36E-01	2.33E+00	5.13E-01	2.09E+00	5.41E-01	1.45E+00	5.48E-01
f_9	7.84E-01	2.27E-01	3.31E+00	5.80E-01	1.99E+00	4.75E-01	1.48E+00	4.36E-01	9.84E-01	2.27E-01
f_{10}	5.73E+01	1.43E+01	2.93E+02	9.09E+01	1.65E+02	5.89E+01	1.19E+02	4.20E+01	7.97E+01	2.20E+01
f_{11}	2.36E-02	3.50E-02	1.01E-01	6.12E-02	5.57E-02	3.51E-02	4.35E-02	3.16E-02	3.28E-02	4.16E-02
f_{12}	1.74E+00	2.45E-02	7.60E+00	7.95E+00	6.68E+00	2.88E+00	5.16E+00	2.57E+00	2.75E+00	1.23E+00

Gerçekleştirilen bu uygulamadan iki farklı inceleme yapılmaktadır. a) PSO ve önerilen IPSO algoritmalarının çözüm kaliteleri açısından karşılaştırılması, b) fine-grained paralel modeli gerçekleştirilen IPSO algoritmasının farklı komşuluk topolojilerinde performansının incelenmesi.

Tablo 5.5'de PSO ve IPSO algoritmaları ile elde edilen sonuçlar incelendiğinde, IPSO algoritmasının çözüm kalitesinin PSO algoritmasının çözüm kalitesine oranla test fonksiyonlarının tamamında daha iyi olduğu görülmüştür. Sonuçlar önerilen modelin PSO algoritmasına bir avantaj kazandırdığını göstermektedir. Tablo 5.6'daki IPSO algoritmasının fine-grained paralel modelinin farklı komşuluk topolojileri ile elde edilen sonuçları incelendiğinde IPSO ve PIPSO algoritmalarının çözüm kalitelerinin birbirlerine yakın olduğu söylenebilir. Farklı komşuluk topolojileri birbirleri ile kıyaslandığında, sonuçlar birbirine yakın olmakla beraber Compact25 komşuluk topolojisinde daha geniş bir komşu kümesi içermesi sebebi ile diğerlerine nazaran ortalama kalitenin biraz daha iyi olduğu söylenebilir. Ancak Compact25 komşuluk topolojisinde işlemciler arasında çok fazla iletişim gerçekleştiği dikkate alınmalıdır.

5.4. ABC Algoritmasının Coarse-Grained Paralel Gerçekleştirimi ve Parametre Analizi

Bu uygulama da ABC algoritmasının coarse-grained paralelleştirme modelinin detaylı incelenmesi amaçlanmıştır. Genel olarak uyarlaması kolay ve verimliliğin yüksek olduğu bu model popülasyonun alt popülasyonlara bölünmesi ve bazı kriterlere göre bu alt popülasyonların birbirleri ile haberleşmesi prensibi üzerine kuruludur. Coarse-grained paralelleştirme modelinde ki ek yeni parametreler (göç topolojisi, göç stratejileri, göç sıklığı ve göç eden birey sayısı) algoritmaların performanslarını, yakınsama hızlarını ve çalışma zamanlarını etkilemektedir. Bu çalışmada bu parametrelerin gerçekleştirilen paralel ABC (Parallel Artificial Bee Colony, PABC) algoritmasına etkisi analiz edilerek, optimum değerlerinin bulunması amaçlanmıştır.

Yapılan çalışma da, CEC2008 için özel olarak hazırlanmış zorlaştırılmış altı adet test fonksiyonu ile yaygın olarak kullanılan altı adet test fonksiyonu kullanılmıştır.

Test fonksiyonları için problem boyutları 100, 500 ve 1000 değerleri için ayrı ayrı alınırken iterasyon sayısı Dx5000 olacak şekilde ayarlanmıştır. Bu değerler CEC2008 yarışması için tanımlanan özel değerlerdir. Ayrıca algoritmaya özgü parametre olan "limit" değeri 2000 olarak seçilmiştir. Çalışmalarda popülasyon boyutu 60 ve 120 olmak üzere iki farklı durum için incelenmiştir. Seri ABC algoritması tek bir işlemci üzerinde çalışırken, PABC algoritmasında her bir alt popülasyon farklı bir işlemci üzerinde çalışmaktadır. Dolayısı ile bu modelde işlemci sayısı ve alt popülasyon sayısı birbirine eşittir. Alt popülasyonlardaki birey sayısı Eşitlik (5.1) ile hesaplanmıştır.

$$\mathrm{NP}_{\mathrm{altpopulasyon}} = \frac{\mathrm{NP}}{\text{işlemci sayısı}} \tag{5.1}$$

PABC algoritmasının performans ve çalışma zamanı analizleri için; alt popülasyon sayısı 20, 15, 10, 5, 4, 3 ve 2 değerleri için, göç sıklığı 10, 20, 30, 40, 50, 100, 250, 500, 1000 ve 2500 değerleri için, göç topolojisi halka, halka+1+2, küp ve grid çeşitleri için incelenmiştir. Gerçekleştirilen uygulamalarda göç büyüklüğü 1 olarak alınırken, göç stratejisi olarak en iyi bireyin göç ederek en kötü bireyle yer değiştirdiği strateji benimsenmiştir.

5.4.1. Popülasyon Büyüklüğü ve Alt Popülasyon Sayısının Çözüm Kalitesine Etkisi

Alt popülasyon sayısının etkisi; popülasyon büyüklüğü 120 olmak üzere D=100 için Tablo 5.7, D=500 için Tablo 5.8 ve D=1000 için Tablo 5.9, popülasyon büyüklüğü 60 olmak üzere D=100 için Tablo 5.10, D=500 için Tablo 5.11 ve D=1000 için Tablo 5.12'de 30 koşma sonucu oluşan çözümlerin ortalama uygunluk ve standart sapma değerleri verilerek gösterilmiştir. Tablolarda en iyi çözüm sonuçları kalın font ile verilmiştir. Tablo 5.7, Tablo 5.8, Tablo 5.9, Tablo 5.10, Tablo 5.11 ve Tablo 5.12'de verilen sonuçlarda halka topolojisi kullanılmıştır, göç sıklığı 10 alınmıştır.

Sonuçların istatistiksel olarak karşılaştırılması amacı ile ilk olarak verilerin normalliğini test eden Kolmogorov Smirnov testi uygulanmıştır [163]. Test sonucunun negatif çıkması ile verilerin normal bir dağılımdan gelmediği görülmüştür.

Normal dağılıma uymayan verilerin birbirleri ile karşılaştırılması için yaygın olarak kullanılan Wilcoxon testi, verilerin aynı ana kütleye ait olup olmadığını analiz etmek için kullanılan bir yöntemdir [164]. Bu amaçla Tablo 5.7, Tablo 5.8 ve Tablo 5.9'daki sonuçların istatistiksel karşılaştırmaları Tablo 5.13'de, Tablo 5.10, Tablo 5.11, Tablo 5.12'deki sonuçların istatistiksel karşılaştırmaları Tablo 5.14'de verilmiştir. Çift yönlü Wilcoxon testi için anlamlılık düzeyi=0.05 alınırken, Tablo 5.13 ve Tablo 5.14'deki sıfır değerleri karşılaştırma yapılan iki veri arasında istatistiksel olarak anlamlı bir fark olmadığını, ok işaretleri ise karşılaştırma yapılan iki veri arasında istatistiksel olarak anlamlı bir fark olduğunu göstermektedir. Aynı zamanda tablolardaki ok yönleri başarılı olan algoritmayı göstermektedir.

Hem ABC hem de PABC algoritmalarında popülasyon büyüklüğünün performansa etkisi yorumlandığında; NP=60 ile elde edilen sonuçların, NP=120 ile elde edilen sonuçlardan daha iyi olduğu görülmektedir. Bu sonuçlar ABC ve PABC algoritmalarının düşük popülasyonda daha başarılı olduğunu göstermektedir. ABC ve PABC algoritmaları karşılaştırıldığında ise PABC algoritmasının hem NP=60 hem de NP=120 de ABC algoritmasından daha iyi olduğu görülmektedir. 0.05 anlamlılık düzeyinde çift yönlü Wilcoxon testi, ABC ve PABC algoritmalarının f_2, f_3, f_4, f_6, f_8 ve f_{10} fonksiyon çözümlerinde istatistiksel olarak farklı olduğunu, f_1, f_5, f_7, f_9, f_{11} ve f_{12} fonksiyon çözümlerinde ise istatistiksel olarak farklı olmadığını göstermiştir. Ayrıca, alt popülasyon sayısının artmasıyla PABC algoritmasının performansının da arttığı gözlemlenmiştir. PABC algoritmasındaki bu performans iyileşmesi test fonksiyonlarına göre farklılık göstermektedir. Tablo 5.7, Tablo 5.8, Tablo 5.9, Tablo 5.10, Tablo 5.11 ve Tablo 5.12'de PABC algoritması ABC algoritmasına kıyasla tüm test fonksiyonlarında daha iyi ya da eşit sonuçlar vermiştir. Özellikle Rastrigin fonksiyonunda PABC algoritmasının başarı oranı oldukça yüksek çıkmıştır. PABC algoritmasında iyi bireylerin alt popülasyonlara transferleri ile algoritmanın yakınsama hızı gelişmektedir. İyi bireylerin daha fazla sayıda iyileştirilmeye çalışılması; en iyi çözümü bulmak için daha fazla iterasyona ihtiyaç duyan çok modlu Rastrigin fonksiyonu için çözümü oldukça etkilemektedir.

Tablo 5.7. ABC ve farklı CPU sayıları ile koşulan PABC algoritmalarının ortalama ve standart sapma değerleri, Ort.: En iyi değerlerin ortalamaları, Std.: En iyi değerlerin standart sapmaları ($NP = 120$, $D = 100$)

	Fonksiyon	f_1			f_2			f_3			f_4		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	1	**0**	0	1.1464	8.48E+01	2.79E+00	1.0204	1.06E+01	8.39E+00	1.5892	2.01E+00	8.93E-01	5.5356
PABC	2	**0**	0	0.9156	7.13E+01	9.72E+00	0.8472	9.74E+00	9.75E+00	1.1712	3.58E-01	5.54E-01	3.098
	3	**0**	0	0.5944	5.25E+01	1.66E+01	0.5404	1.19E+01	1.37E+01	0.7524	3.67E-09	3.57E-09	2.0416
	4	**0**	0	0.4376	4.81E+01	1.45E+01	0.3984	1.17E+01	1.12E+01	0.5504	4.36E-07	9.56E-06	1.5192
	5	**0**	0	0.3532	3.48E+01	1.63E+01	0.32	8.58E+00	6.72E+00	0.4476	1.07E-09	2.93E-09	1.2144
	10	**0**	0	0.1764	5.27E+00	8.31E+00	0.1596	8.73E+00	8.66E+00	0.2228	5.54E-10	1.24E-09	0.6092
	15	**0**	0	0.12	2.87E+00	8.27E+00	0.1072	4.20E+00	1.42E+01	0.1484	4.57E-10	1.93E-09	0.3972
	20	**0**	0	0.0912	**2.14E+00**	6.42E+00	0.0828	**3.12E-02**	1.45E-01	0.1144	**1.18E-10**	4.01E-10	0.2964
	Fonksiyon	f_5			f_6			f_7			f_8		
	CPU Sayısı)	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	1	**0**	0	10.5568	1.25E-06	1.18E-06	5.5372	**0**	0	1.0312	3.46E-01	4.64E-02	8.3464
PABC	2	**0**	0	5.7032	5.93E-07	1.96E-07	3.0928	**0**	0	0.848	3.05E-01	7.67E-02	5.6348
	3	**0**	0	3.7648	5.35E-07	2.01E-07	2.0544	**0**	0	0.5504	2.83E-01	5.15E-02	3.1896
	4	**0**	0	2.806	4.85E-07	2.84E-07	1.5232	**0**	0	0.4088	2.46E-01	6.88E-02	2.47
	5	**0**	0	2.2788	6.06E-07	3.53E-07	1.2292	**0**	0	0.322	2.46E-01	7.52E-02	2.224
	10	**0**	0	1.1536	3.84E-07	3.55E-07	0.6104	**0**	0	0.1612	1.46E-01	5.51E-02	1.0632
	15	**0**	0	0.7756	8.14E-08	1.20E-07	0.4016	**0**	0	0.1104	8.14E-02	5.08E-02	0.6508
	20	**0**	0	0.5704	**8.89E-09**	1.74E-08	0.2916	**0**	0	0.0832	**6.48E-02**	3.45E-02	0.55
	Fonksiyon	f_9			f_{10}			f_{11}			f_{12}		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	1	7.92E-09	5.43E-09	0.9948	4.04E-03	2.40E-03	1.064	**0**	0	10.558	**0**	0	9.1788
PABC	2	7.15E-09	3.26E-09	0.89	3.52E-03	2.61E-03	0.9264	**0**	0	5.7144	**0**	0	5.0472
	3	7.52E-09	2.72E-09	0.5816	**3.20E-03**	2.41E-03	0.6036	**0**	0	3.8104	**0**	0	3.3588
	4	7.40E-09	2.92E-09	0.4304	2.29E-02	9.76E-02	0.4472	**0**	0	2.8772	**0**	0	2.5148
	5	6.42E-09	3.53E-09	0.3376	4.59E-03	7.10E-03	0.3552	**0**	0	2.2964	**0**	0	2.0336
	10	8.53E-09	7.09E-09	0.1672	3.70E-01	2.04E-01	0.176	**0**	0	1.2488	**0**	0	1.0904
	15	1.47E-09	1.86E-09	0.114	8.29E-02	1.82E-01	0.1216	**0**	0	0.7868	**0**	0	0.6956
	20	**2.20E-10**	3.22E-10	0.0872	3.23E-01	2.36E-01	0.0932	**0**	0	0.6092	**0**	0	0.5188

Tablo 5.8. ABC ve farklı CPU sayıları ile koşulan PABC algoritmalarının ortalama ve standart sapma değerleri, Ort.: En iyi değerlerin ortalamaları, Std.: En iyi değerlerin standart sapmaları ($NP = 120$, $D = 500$)

	Fonksiyon	f_1			f_2			f_3			f_4		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	1	**0**	0	25.9008	1.47E+02	4.91E+00	22.5136	2.94E+01	1.75E+01	37.0484	3.35E+00	1.91E+00	135.7208
PABC	2	**0**	0	21.7876	1.29E+02	6.99E+00	19.8284	2.25E+01	1.77E+01	27.4868	4.41E-01	1.06E+00	86.298
	3	**0**	0	13.1108	1.16E+02	1.34E+01	11.8676	3.25E+01	2.81E+01	16.9996	7.96E-02	2.70E-01	49.4464
	4	**0**	0	9.9664	1.05E+02	1.65E+01	8.99	3.50E+01	2.58E+01	12.6888	2.95E-09	2.24E-09	37.892
	5	**0**	0	7.846	9.18E+01	2.04E+01	7.0304	4.62E+01	3.72E+01	10.0952	2.46E-09	1.81E-09	29.7524
	10	**0**	0	3.8996	2.40E+01	2.70E+01	3.4888	3.34E+01	3.80E+01	5.0056	4.51E-09	5.16E-09	14.7192
	15	**0**	0	2.6416	8.45E+00	1.71E+01	2.4052	1.96E+01	3.80E+01	3.396	3.36E-09	8.37E-09	9.7228
	20	**0**	0	2.002	**8.14E+00**	1.73E+01	1.7796	**1.63E+01**	4.42E+01	2.5332	**1.94E-09**	9.50E-09	7.0936
	Fonksiyon	f_5			f_6			f_7			f_8		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	1	**0**	0	258.2264	2.12E-06	1.43E-06	134.3148	**0**	0	20.7136	1.99E+00	1.85E-01	210.0348
PABC	2	**0**	0	162.2008	1.15E-06	3.67E-07	81.9508	**0**	0	17.4424	1.27E+00	2.43E-01	119.1044
	3	**0**	0	94.5052	9.93E-07	4.72E-07	49.3076	**0**	0	11.9344	1.11E+00	2.43E-01	77.9584
	4	**0**	0	69.7668	9.23E-07	4.21E-07	36.6544	**0**	0	8.7572	8.96E-01	3.45E-01	64.4848
	5	**0**	0	55.4632	9.06E-07	5.03E-07	29.3552	**0**	0	6.8312	9.28E-01	2.75E-01	48.6756
	10	**0**	0	27.926	6.06E-07	7.72E-07	14.6476	**0**	0	3.4516	6.55E-01	3.22E-01	27.8716
	15	**0**	0	19.7424	4.55E-07	1.24E-06	9.7388	**0**	0	2.3228	3.49E-01	2.86E-01	19.276
	20	**0**	0	14.3176	**6.28E-08**	2.10E-07	7.094	**0**	0	1.7616	**2.16E-01**	2.34E-01	13.2276
	Fonksiyon	f_9			f_{10}			f_{11}			f_{12}		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	1	3.93E-08	7.89E-09	21.8872	9.06E+00	1.45E+01	23.0052	**0**	0	262.3164	**0**	0	225.3816
PABC	2	6.05E-08	1.17E-08	18.2272	1.09E+01	1.30E+01	19.1668	**0**	0	140.894	**0**	0	121.4256
	3	4.75E-08	1.57E-08	12.1964	2.06E+01	2.35E+01	12.9116	**0**	0	93.8384	**0**	0	81.7076
	4	4.55E-08	1.85E-08	9.0944	2.20E+01	1.96E+01	9.5096	**0**	0	71.0836	**0**	0	60.7684
	5	4.60E-08	2.39E-08	7.2616	3.12E+01	3.04E+01	7.572	**0**	0	56.25	**0**	0	48.83
	10	8.71E-08	8.46E-08	3.628	1.32E+00	2.76E+00	3.8616	**0**	0	29.692	1.47E-13	1.95E-13	26.0992
	15	1.63E-08	2.46E-08	2.4428	9.94E+00	2.70E+01	2.582	**0**	0	19.3	**0**	0	16.2668
	20	**1.60E-09**	2.44E-09	1.8376	**5.00E-01**	1.33E-07	1.9376	**0**	0	14.7284	**0**	0	12.2462

Tablo 5.9. ABC ve farklı CPU sayıları ile koşulan PABC algoritmalarının ortalama ve standart sapma değerleri, Ort.: En iyi değerlerin ortalamaları, Std.: En iyi değerlerin standart sapmaları ($NP = 120$, $D = 1000$)

	Fonksiyon	f_1			f_2			f_3			f_4		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	1	**0**	0	104.3556	1.74E+02	1.76E+00	90.1604	5.96E+01	2.43E+01	151.7932	3.34E+00	1.78E+00	552.8196
PABC	2	**0**	0	77.828	1.45E+02	8.89E+00	70.7512	6.00E+01	3.55E+01	102.2572	4.98E-01	1.32E+00	298.8188
	3	**0**	0	51.99	1.41E+02	6.78E+00	46.8096	4.53E+01	4.54E+01	68.1604	1.02E-08	1.85E-08	200.8688
	4	**0**	0	39.746	1.27E+02	1.24E+01	35.592	7.16E+01	5.16E+01	50.4792	7.16E-09	1.39E-08	148.1864
	5	**0**	0	31.4548	1.14E+02	1.88E+01	28.1272	9.68E+01	6.38E+01	40.7664	5.95E-09	1.28E-08	120.3636
	10	**0**	0	15.756	4.57E+01	4.05E+01	14.034	5.64E+01	6.15E+01	20.152	**1.60E-09**	2.02E-09	58.0448
	15	**0**	0	10.6624	1.35E+01	2.22E+01	9.5548	1.13E+01	5.40E+01	13.5592	2.34E-09	5.52E-09	38.3532
	20	**0**	0	8.1292	**1.19E+01**	2.14E+01	7.3128	**6.78E+00**	2.97E+01	10.4288	1.84E-09	8.99E-09	28.9712
	Fonksiyon	f_5			f_6			f_7			f_8		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	1	**0**	0	1034.535	6.97E-06	8.37E-06	551.6256	**0**	0	81.8068	4.28E+00	4.87E-01	900.8444
PABC	2	**0**	0	564.8704	1.81E-06	8.62E-07	301.07	**0**	0	70.124	2.38E+00	1.63E-01	483.6008
	3	**0**	0	382.3564	1.41E-06	3.72E-07	198.7972	**0**	0	45.712	1.90E+00	5.06E-01	343.7648
	4	**0**	0	282.3304	1.24E-06	5.30E-07	148.2776	**0**	0	34.31	1.91E+00	4.43E-01	249.7428
	5	**0**	0	221.136	1.12E-06	6.09E-07	119.3652	**0**	0	27.7984	1.58E+00	4.63E-01	217.86
	10	**0**	0	114.2372	1.31E-06	1.21E-06	60.4572	**0**	0	13.7496	1.43E+00	5.32E-01	122.4044
	15	**0**	0	76.7292	3.61E-07	8.72E-07	39.9092	**0**	0	9.1996	1.05E+00	6.24E-01	86.856
	20	**0**	0	58.6756	**2.67E-07**	8.74E-07	29.2816	**0**	0	6.924	**4.24E-01**	4.08E-01	70.1292
	Fonksiyon	f_9			f_{10}			f_{11}			f_{12}		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	1	9.10E-08	2.44E-08	86.6232	1.94E+02	8.14E+01	92.2916	**0**	0	1047.9415	**0**	0	901.0845
PABC	2	1.27E-07	2.65E-08	72.6752	2.14E+02	8.51E+01	75.8212	**0**	0	566.2028	**0**	0	488.9364
	3	2.84E-07	5.39E-07	48.2096	2.27E+02	8.58E+01	50.6492	**0**	0	383.3388	**0**	0	325.41
	4	1.13E-07	3.08E-08	36.2532	2.30E+02	9.95E+01	37.9188	**0**	0	287.6076	**0**	0	251.004
	5	1.12E-07	3.80E-08	29.6628	2.21E+02	1.12E+02	31.9036	**0**	0	228.7836	**0**	0	195.9408
	10	1.21E-07	9.94E-08	14.4292	1.41E+01	2.88E+01	15.0328	**0**	0	120.9584	2.09E-13	2.91E-13	106.9828
	15	2.35E-08	3.17E-08	9.8888	1.66E+02	2.01E+02	10.3936	**0**	0	79.468	**0**	0	66.6856
	20	**3.75E-09**	6.61E-09	7.416	**5.00E-01**	2.30E-07	7.7272	**0**	0	60.6864	**0**	0	49.5788

Tablo 5.10. ABC ve farklı CPU sayıları ile koşulan PABC algoritmalarının ortalama ve standart sapma değerleri, Ort.: En iyi değerlerin ortalamaları, Std.: En iyi değerlerin standart sapmaları ($NP = 60$, $D = 100$)

	Fonksiyon	f_1			f_2			f_3			f_4		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	1	**0**	0	1.1184	5.80E+01	5.39E+00	1.0036	9.40E+00	1.19E+01	1.542	**0**	0	4.9792
PABC	2	**0**	0	0.846	4.18E+01	1.46E+01	0.7996	4.94E+00	5.48E+00	1.0728	**0**	0	2.8032
	3	**0**	0	0.5576	2.43E+01	1.34E+01	0.5252	1.20E+01	1.69E+01	0.7092	**0**	0	1.8784
	4	**0**	0	0.4796	1.26E+01	8.53E+00	0.8548	**4.64E+00**	6.15E+00	0.7744	**0**	0	1.4248
	5	**0**	0	0.3392	5.45E+00	6.41E+00	0.3188	1.13E+01	1.50E+01	0.4448	**0**	0	1.2396
	10	**0**	0	0.1748	**8.94E-01**	2.06E+00	0.1636	6.75E+00	1.58E+01	0.2236	**0**	0	0.5924
	Fonksiyon	f_5			f_6			f_7			f_8		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	1	**0**	0	10.51	**0**	0	4.5296	**0**	0	0.9296	2.09E-01	2.52E-02	8.4588
PABC	2	**0**	0	5.8332	**0**	0	2.5392	**0**	0	0.8024	1.77E-01	4.40E-02	4.5992
	3	**0**	0	3.9284	**0**	0	1.6996	**0**	0	0.5336	1.61E-01	3.30E-02	3.1068
	4	**0**	0	4.2468	**0**	0	2.5956	**0**	0	0.3956	1.38E-01	4.92E-02	2.37
	5	**0**	0	2.3408	**0**	0	1.0272	**0**	0	0.3176	1.21E-01	4.88E-02	1.9068
	10	**0**	0	1.1968	**0**	0	0.5032	**0**	0	0.1616	**6.01E-02**	3.50E-02	1.2704
	Fonksiyon	f_9			f_{10}			f_{11}			f_{12}		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	1	**0**	0	0.9632	6.33E-06	2.82E-06	1.0328	**0**	0	10.4424	**0**	0	9.042
PABC	2	**0**	0	0.8392	3.60E-06	2.82E-06	0.8736	**0**	0	6.5288	**0**	0	5.7052
	3	**0**	0	0.5572	**2.34E-06**	1.56E-06	0.5796	**0**	0	4.3708	**0**	0	3.7984
	4	**0**	0	0.4092	5.76E-06	5.42E-06	0.4328	**0**	0	3.226	**0**	0	2.7852
	5	**0**	0	0.3256	2.43E-05	1.02E-04	0.348	**0**	0	2.6572	**0**	0	2.3164
	10	**0**	0	0.1676	1.38E-05	3.12E-05	0.178	**0**	0	1.4612	**0**	0	1.2944

Tablo 5.11. ABC ve farklı CPU sayıları ile koşulan PABC algoritmalarının ortalama ve standart sapma değerleri, Ort.: En iyi değerlerin ortalamaları, Std.: En iyi değerlerin standart sapmaları ($NP = 60$, $D = 500$)

	Fonksiyon	f_1			f_2			f_3			f_4		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	1	**0**	0	25.4808	1.48E+02	4.18E+00	22.3832	7.17E+00	1.11E+01	36.2892	1.44E-02	6.83E-02	123.018
PABC	2	**0**	0	18.8356	1.08E+02	1.05E+01	17.4048	9.41E+00	1.57E+01	24.9596	**0**	0	67.4912
	3	**0**	0	12.546	8.29E+01	2.16E+01	11.5824	**7.07E+00**	8.85E+00	16.6152	**0**	0	45.6104
	4	**0**	0	15.4088	6.93E+01	2.47E+01	11.1008	1.23E+01	1.68E+01	16.3212	**0**	0	36.9488
	5	**0**	0	7.918	5.05E+01	2.76E+01	7.028	9.88E+00	1.58E+01	10.1036	**0**	0	28.0732
	10	**0**	0	4.0148	**4.70E+00**	1.01E+01	3.6264	1.35E+01	3.42E+01	5.0404	**0**	0	14.8504
	Fonksiyon	f_5			f_6			f_7			f_8		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	1	**0**	0	258.3985	**0**	0	109.2296	**0**	0	20.5644	1.34E+00	9.99E-02	216.7608
PABC	2	**0**	0	152.8	**0**	0	61.4892	**0**	0	17.0164	8.23E-01	2.27E-01	144.8392
	3	**0**	0	102.6984	**0**	0	41.4908	**0**	0	11.3348	6.09E-01	2.04E-01	79.8152
	4	**0**	0	76.3732	**0**	0	31.3644	**0**	0	8.4472	6.22E-01	2.42E-01	68.0408
	5	**0**	0	61.7716	**0**	0	24.82	**0**	0	6.8244	4.59E-01	2.33E-01	53.5332
	10	**0**	0	31.6432	**0**	0	12.682	**0**	0	3.5048	**2.99E-01**	1.95E-01	27.9042
	Fonksiyon	f_9			f_{10}			f_{11}			f_{12}		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	1	**0**	0	21.2448	1.13E+01	1.26E+01	22.5864	**0**	0	262.0268	**0**	0	224.742
PABC	2	**0**	0	18.1712	1.01E+01	1.78E+01	19.1028	**0**	0	170.5044	**0**	0	146.4996
	3	**0**	0	11.8512	1.11E+01	1.59E+01	12.5092	**0**	0	112.8336	**0**	0	97.9168
	4	**0**	0	8.9452	1.34E+01	2.05E+01	9.432	**0**	0	83.5308	**0**	0	71.7384
	5	**0**	0	7.1892	1.53E+01	2.11E+01	7.6388	**0**	0	68.3536	**0**	0	59.0912
	10	**0**	0	3.6464	**6.09E-02**	1.62E-01	3.9388	**0**	0	36.8312	**0**	0	32.3021

Tablo 5.12. ABC ve farklı CPU sayıları ile koşulan PABC algoritmalarının ortalama ve standart sapma değerleri, Ort.: En iyi değerlerin ortalamaları, Std.: En iyi değerlerin standart sapmaları ($NP = 60$, $D = 1000$)

	Fonksiyon	f_1			f_2			f_3			f_4		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	1	**0**	0	103.48	1.74E+02	1.68E+00	90.2932	3.68E+00	1.19E+01	146.9744	1.14E-01	3.09E-01	491.8496
PABC	2	**0**	0	75.4668	1.34E+02	9.02E+00	70.3208	**2.58E+00**	3.15E+00	101.3628	**0**	0	273.1324
	3	**0**	0	49.9508	1.19E+02	1.25E+01	46.176	3.69E+00	3.42E+00	66.6512	5.73E-09	2.80E-08	186.328
	4	**0**	0	50.4108	1.07E+02	1.34E+01	36.7624	9.63E+00	1.87E+01	51.1048	**0**	0	142.806
	5	**0**	0	30.3428	8.04E+01	2.53E+01	27.88	1.71E+01	2.80E+01	40.0144	**0**	0	115.6564
	10	**0**	0	15.4132	**1.67E+01**	2.28E+01	14.2192	3.07E+00	1.49E+01	20.4388	**0**	0	58.8032
	Fonksiyon	f_5			f_6			f_7			f_8		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	1	**0**	0	1028.254	1.42E-11	5.06E-13	441.0643	**0**	0	81.0272	3.07E+00	2.44E-01	870.934
PABC	2	**0**	0	597.2936	1.19E-11	4.64E-13	251.434	**0**	0	67.7736	1.59E+00	5.86E-01	536.0856
	3	**0**	0	409.9897	1.12E-11	4.01E-13	166.0632	**0**	0	46.1768	1.35E+00	3.56E-01	381.1568
	4	**0**	0	304.1204	1.08E-11	6.65E-13	127.684	**0**	0	34.2916	1.24E+00	2.82E-01	238.4484
	5	**0**	0	250.3748	**0**	0	102.6228	**0**	0	27.9248	1.04E+00	4.09E-01	209.8348
	10	**0**	0	125.768	**0**	0	51.3828	**0**	0	13.5988	**7.38E-01**	4.49E-01	128.9576
	Fonksiyon	f_9			f_{10}			f_{11}			f_{12}		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	1	**0**	0	84.1308	7.01E+01	5.26E+01	89.4792	**0**	0	1039.5071	**0**	0	896.1972
PABC	2	**0**	0	71.1132	1.48E+02	6.42E+01	74.5208	**0**	0	686.3208	1.95E-13	6.69E-13	612.3076
	3	3.95E-08	1.39E-07	47.6668	1.47E+02	8.08E+01	49.9952	**0**	0	457.2236	**0**	0	408.4172
	4	2.45E-13	9.26E-13	36.0776	1.49E+02	9.88E+01	37.9764	**0**	0	342.8152	**0**	0	295.2672
	5	**0**	0	29.4784	1.73E+02	1.10E+02	30.7008	**0**	0	282.594	**0**	0	249.058
	10	**0**	0	14.2608	**2.01E-01**	2.44E-01	15.2404	**0**	0	147.7784	**0**	0	130.0928

Tablo 5.13. ABC ve farklı CPU sayıları ile koşulan PABC algoritmalarının istatistiksel karşılaştırılmaları (NP=120, Çift yönlü Wilcoxon testi, Anlamlılık düzeyi=0.05)

			PABC											
			CPU Sayısı											
			20			15			10			5		
Algoritma	CPU Sayısı	Fonksiyon	100	500	1000	100	500	1000	100	500	1000	100	500	1000
ABC	1	f_2	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
		f_3	↑	↑	↑	↑	↑	↑	0	0	0	0	0	←
		f_4	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
		f_6	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
		f_8	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
		f_{10}	←	0	↑	←	0	0	←	0	↑	0	←	0
PABC	20	f_2				←	0	0	←	←	←	←	←	←
		f_3				←	←	←	←	←	←	←	←	←
		f_4				←	←	←	←	←	↑	←	←	←
		f_6				←	←	←	←	←	←	←	←	←
		f_8				0	←	←	←	←	←	←	←	←
		f_{10}				0	0	←	←	←	0	↑	←	←
	15	f_2							←	←	←	←	←	←
		f_3							←	←	←	←	←	←
		f_4							←	←	↑	←	↑	←
		f_6							←	←	←	←	←	←
		f_8							←	←	←	←	←	←
		f_{10}							←	0	↑	↑	←	←
	10	f_2										←	←	←
		f_3										0	0	←
		f_4										←	0	←
		f_6										←	←	0
		f_8										←	←	0
		f_{10}										↑	←	←

Tablo 5.14. ABC ve farklı CPU sayıları ile koşulan PABC algoritmalarının istatistiksel karşılaştırılmaları (NP=60, Çift yönlü Wilcoxon testi, Anlamlılık düzeyi=0.05)

			PABC											
			CPU Sayısı											
			10			5			4			3		
Algoritma	CPU Sayısı	Fonksiyon	100	500	1000	100	500	1000	100	500	1000	100	500	1000
ABC	1	f_2	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
		f_3	↑	←	↑	0	0	←	0	0	←	0	0	←
		f_4	0	↑	↑	0	↑	↑	0	↑	↑	0	↑	↑
		f_6	0	0	↑	0	0	↑	0	0	↑	0	0	↑
		f_8	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
		f_{10}	←	↑	↑	←	0	←	0	0	←	↑	0	←
PABC	10	f_2				←	←	←	←	←	←	←	←	←
		f_3				←	↑	←	↑	↑	←	←	↑	←
		f_4				0	0	0	0	0	0	0	0	←
		f_6				0	0	←	0	0	←	0	0	←
		f_8				←	←	←	←	←	←	←	←	←
		f_{10}				0	←	←	0	←	←	0	←	←
	5	f_2							←	←	←	←	←	←
		f_3							0	0	0	0	0	0
		f_4							0	0	0	0	0	←
		f_6							0	0	←	0	0	←
		f_8							0	←	←	←	←	←
		f_{10}							0	0	0	0	0	0
	4	f_2										←	←	←
		f_3										0	0	0
		f_4										0	0	←
		f_6										0	0	0
		f_8										0	0	0
		f_{10}										↑	0	0

5.4.2. Göç Sıklığının Çözüm Kalitesine Etkisi

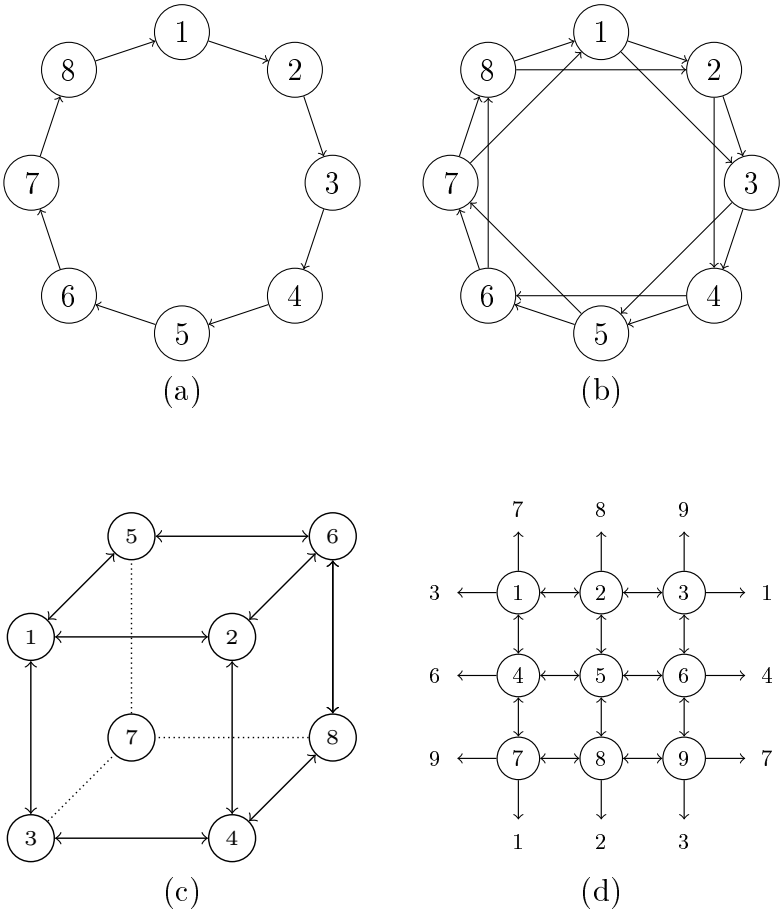
Göç sıklığının PABC algoritmasının performansına etkisi için çalışmada kullanılan tüm test fonksiyonları tüm problem boyutlarında incelenmiştir. Benzer sonuçlar alındığı için sadece Tablo 5.15'de f_4 fonksiyonun D=1000 boyutlu çözümü ve Tablo 5.16'da f_6 fonksiyonun D=500 boyutlu çözümü verilmiştir. Tablolarda 20/15/10/5/4/3 sütunları alt populasyon sayısını, satırlarda göç sıklık değerini göstermektedir. Ayrıca alt popülasyonlar arası hiç göç gerçekleşmediğinde elde edilen sonuçlarda göç sıklığı satırında "Göç yok" şeklinde gösterilmiştir. Bu tablolarda verilen sonuçlarda halka topolojisi kullanılmış ve popülasyon büyüklüğü NP=120 alınmıştır.

Tablo 5.15 ve Tablo 5.16, alt popülasyonlar arası düşük aralıklarla yapılan göç işleminin f_4 ve f_6 fonksiyonlarının performansını iyileştirdiğini, yüksek aralıklarla yapılan göç işleminin, kaliteli çözümlerin dağılımını yavaşlattığını ve algoritmanın performansına yeterince katkı sağlayamadığını göstermektedir. Ayrıca alt popülasyonlar arası göç işlemi hiç olmadığı zaman, bunun algoritmaya hiç bir katkı sağlamadığı ve küçük popülasyon ile çalışan bir dizi seri algoritmadan farklı olmadığı görülmüştür. Yapılan çalışmalar, PABC algoritması ile daha iyi çözüm kalitesi elde etmek için, alt popülasyonlar arası iletişimin sık olması gerektiğini göstermiştir. Ayrıca, alt popülasyon sayısının artması ile ayarlanmak istenen göç sıklık değerinin öneminin de arttığı görülmektedir. Bu çalışmada gerçekleştirilen deneyler için göç sıklık değeri için 10 değeri tavsiye edilebilir.

5.4.3. Göç Topolojisinin Çözüm Kalitesine Etkisi

Göç topolojisinin PABC algoritmasının performansına etkisini incelemek amacı ile yapılan çalışmada dört farklı topoloji gerçekleştirilmiştir. Gerçekleştirilen topolojiler Şekil 5.3'de grafiksel olarak gösterilen halka (T_1), halka+1+2 (T_2), küp (T_3) ve grid (T_4) topolojileridir. Elde edilen sonuçlar Tablo 5.17'de verilmiştir. Ayrıca gerçekleştirilen farklı topolojilerin algoritmanın yakınsama hızına etkisini

göstermek amacı ile Şekil 5.4 ve Şekil 5.5'de yakınsama grafikleri verilmiştir. Bu çalışmada, gerçekleştirilen tüm topolojilere uygun olacak şekilde, popülasyon büyüklüğü NP=96, alt popülasyon sayısı 16 ve göç sıklık değeri 10 olarak alınmıştır.



Şekil 5.3. Deneysel çalışmalarda incelenenen göç topolojileri, a) Halka, b) Halka+1+2, c) Küp ve d) Grid

Göç topolojileri genellikle coarse-grained paralelleştirme modeli ile gerçekleştirilen paralel algoritmalar için önemli bir bileşendir. Tablo 5.17'deki sonuçlar, Şekil 5.4 ve Şekil 5.5'deki grafikler incelendiğinde PABC algoritması için göç topolojisinin sadece performansı etkilemekle kalmayıp aynı zamanda yakınsama hızlarını da etkilediği görülmektedir. Topolojiler bir birleri ile karşılaştırıldığında ise grid topolojisinin gerçekleştirilen diğer topolojilere nazaran kısmen biraz daha iyi olduğu söylenebilir.

Tablo 5.15. PABC algoritmasının farklı göç sıklık değerleri için f_4 fonksiyonu çözüm sonuçları, Ort.: En iyi değerlerin ortalamaları, Std.: En iyi değerlerin standart sapmaları ($NP = 120$, $D = 1000$)

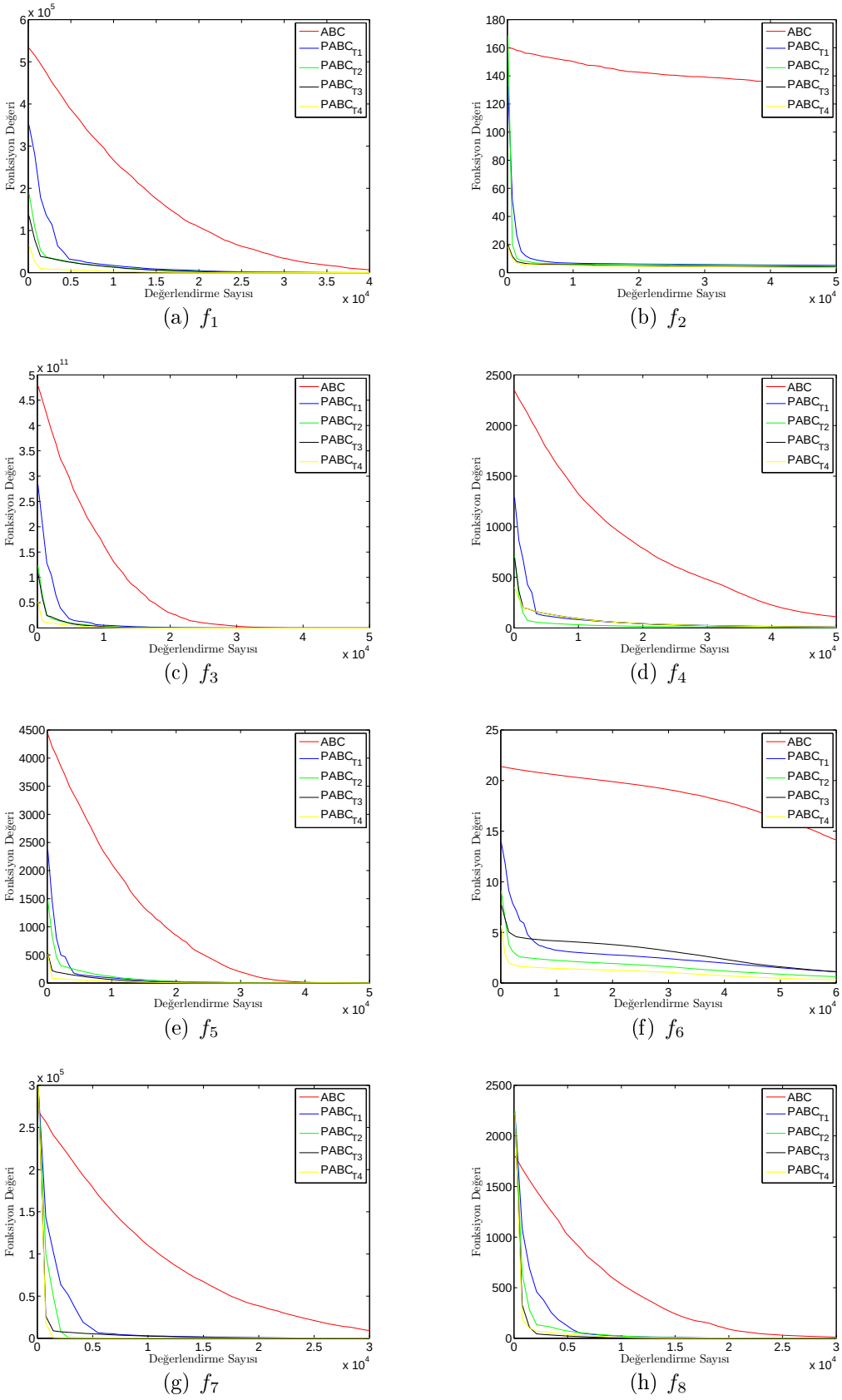
	Alt Popülasyon Sayısı								
	20			15			10		
Göç Sıklığı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
10	**1.84E-09**	8.99E-09	28.9712	**2.34E-09**	5.52E-09	38.3532	**1.60E-09**	2.02E-09	58.0448
20	2.41E-07	8.18E-07	28.7352	2.34E-08	6.50E-08	37.8456	7.70E-09	1.44E-08	57.3692
30	1.24E-06	6.09E-06	28.4864	1.08E-07	3.06E-07	37.4684	1.91E-08	3.93E-08	57.4924
40	1.57E-06	7.71E-06	28.2744	1.93E-07	6.40E-07	37.5928	2.65E-08	4.37E-08	57.4800
50	3.99E-02	1.95E-01	28.1576	7.54E-07	3.70E-06	37.4420	9.62E-08	2.23E-07	57.6540
100	5.48E-04	2.12E-03	28.4180	1.20E-06	4.18E-06	38.2720	5.24E-07	1.19E-06	57.7624
250	2.00E-01	9.81E-01	29.0464	1.79E-01	6.98E-01	38.9188	1.26E-03	5.98E-03	57.8852
500	4.86E-01	2.38E+00	30.1652	8.54E-02	4.18E-01	39.9504	2.11E-01	3.97E-01	57.9680
1000	5.73E-01	2.81E+00	32.1024	2.15E+00	4.25E+00	42.6564	1.08E+00	1.87E+00	59.4780
2500	6.50E-01	3.19E+00	32.5540	8.44E-01	2.87E+00	45.9732	1.14E+00	1.95E+00	60.2776
Göç yok	1.31E+02	3.75E+02	29.2480	5.95E+01	4.82E+02	39.0934	1.59E+02	2.73E+02	58.6302
	5			4			3		
Göç Sıklığı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
10	**5.95E-09**	6.60E-08	120.3636	**7.16E-09**	1.39E-08	148.1864	**1.02E-08**	1.85E-08	200.8688
20	6.79E-09	2.92E-08	119.1304	1.89E-06	9.16E-06	146.2060	2.42E-05	1.18E-04	198.5944
30	9.57E-09	2.63E-08	119.2236	1.10E-06	5.31E-06	146.6260	2.15E-06	1.03E-05	198.5200
40	1.36E-08	5.55E-08	120.2628	4.35E-08	1.38E-07	147.3084	3.98E-02	1.95E-01	199.9364
50	4.80E-08	3.31E-07	118.3836	5.27E-08	1.33E-07	147.0672	6.86E-08	1.19E-07	197.6716
100	7.06E-08	5.78E-07	119.3840	8.45E-06	4.10E-05	147.8084	8.07E-02	2.74E-01	198.8756
250	7.96E-02	9.95E-01	119.7688	2.02E-01	5.62E-01	146.7620	3.06E-01	6.60E-01	197.1824
500	4.74E-01	2.99E+00	119.2164	5.96E-01	9.31E-01	147.2972	6.86E-01	8.48E-01	198.0704
1000	3.19E-01	2.09E+00	119.4588	8.82E-01	1.20E+00	147.0524	8.39E-01	8.76E-01	197.4976
2500	1.82E+00	8.41E+00	118.9416	1.94E+00	2.70E+00	147.3548	2.29E+00	2.05E+00	199.5236
Göç yok	1.29E+01	1.70E+01	118.7964	7.81E+00	1.05E+01	146.1000	4.32E+00	2.02E+00	196.7580

Tablo 5.16. PABC algoritmasının farklı göç sıklık değerleri için f_6 fonksiyonu çözüm sonuçları, Ort.: En iyi değerlerin ortalamaları, Std.: En iyi değerlerin standart sapmaları ($NP = 120$, $D = 500$)

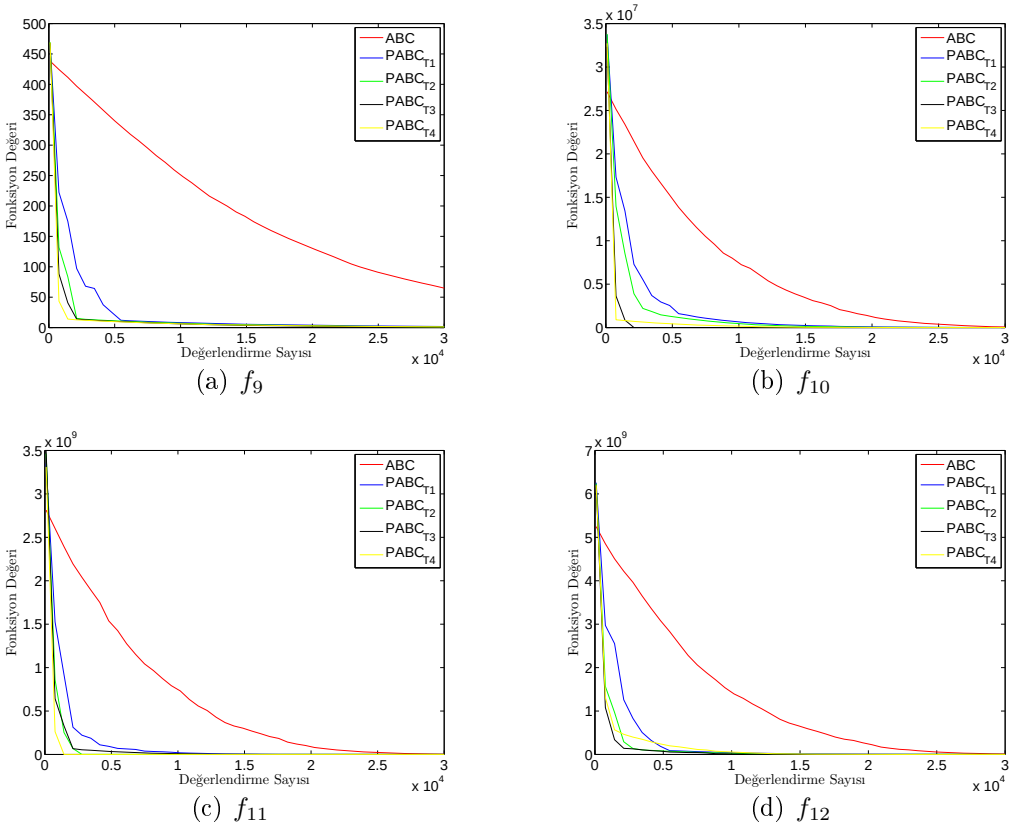
	Alt Popülasyon Sayısı								
	20			15			10		
Göç Sıklığı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
10	**6.28E-08**	2.10E-07	7.0940	**4.55E-07**	1.24E-06	9.7388	**6.06E-07**	7.72E-07	14.6476
20	1.17E-07	3.17E-07	6.9368	8.87E-07	1.74E-06	9.7396	8.51E-07	1.32E-06	14.4500
30	3.30E-07	7.47E-07	6.9544	8.35E-07	1.53E-06	9.5808	2.08E-06	1.91E-06	14.5212
40	1.40E-07	5.67E-07	6.8096	9.27E-07	2.18E-06	9.5944	1.32E-06	1.48E-06	14.4804
50	1.88E-07	8.28E-07	6.7860	7.37E-07	1.56E-06	9.5656	1.51E-06	2.06E-06	14.4156
100	1.21E-07	5.10E-07	6.7364	9.89E-07	1.94E-06	9.4804	1.28E-06	2.18E-06	14.3488
250	1.21E-07	5.40E-07	6.9060	1.06E-06	2.13E-06	9.5500	3.31E-06	3.74E-06	14.4900
500	4.57E-07	2.15E-06	7.0168	2.21E-07	9.08E-07	9.5768	3.08E-06	5.43E-06	14.4760
1000	4.11E-07	1.90E-06	7.3248	1.49E-06	3.31E-06	9.7992	5.69E-06	5.57E-06	14.6160
2500	1.44E-06	6.02E-06	7.7524	8.00E-06	2.12E-05	10.2412	5.11E-06	7.65E-06	14.9104
Göç yok	5.59E-01	9.67E+00	8.0302	2.12E-01	1.49E+00	10.5091	1.75E-01	1.15E+00	15.1580
	5			4			3		
Göç Sıklığı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
10	**9.06E-07**	5.03E-07	29.3552	**9.23E-07**	4.21E-07	36.6544	**9.93E-07**	4.72E-07	49.3076
20	1.61E-06	8.80E-07	29.2772	1.52E-06	8.58E-07	36.7072	1.41E-06	5.63E-07	49.3012
30	2.02E-06	9.47E-07	29.2036	1.87E-06	8.16E-07	36.6532	1.74E-06	4.41E-07	49.2512
40	2.26E-06	1.01E-06	29.1992	1.60E-06	9.32E-07	37.0000	1.84E-06	4.85E-07	49.1928
50	2.24E-06	1.41E-06	29.1212	1.87E-06	7.98E-07	36.5652	2.11E-06	8.48E-07	49.2840
100	3.06E-06	9.08E-07	29.1060	2.37E-06	6.22E-07	36.5388	1.68E-06	8.21E-07	49.1772
250	3.29E-06	1.51E-06	29.4152	2.81E-06	1.08E-06	36.8796	2.01E-06	9.15E-07	49.2316
500	2.91E-06	1.76E-06	29.1460	3.02E-06	1.33E-06	36.5944	2.31E-06	6.66E-07	49.1092
1000	3.46E-06	2.05E-06	29.1812	2.70E-06	8.89E-07	36.6544	2.45E-06	1.11E-06	49.1664
2500	4.16E-06	2.28E-06	29.2688	3.81E-06	2.00E-06	37.3604	3.10E-06	1.74E-06	49.4312
Göç yok	1.23E-05	2.29E-05	29.1385	6.60E-06	1.56E-05	36.5717	3.93E-06	6.92E-06	48.8221

Tablo 5.17. ABC ve farklı göç topolojileri ile koşulan PABC algoritmalarının ortalama ve standart sapma değerleri, Ort.: En iyi değerlerin ortalamaları, Std.: En iyi değerlerin standart sapmaları ($NP = 96$, $D = 100$)

	Fonksiyon	f_1			f_2			f_3		
	Topoloji	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	-	**0**	0	1.0753	7.64E+01	7.16E+00	1.0020	1.03E+01	1.18E+01	1.5357
PABC	T_1	**0**	0	0.1224	1.44E+00	3.32E+00	0.1080	1.55E+00	2.62E+00	0.1409
	T_2	**0**	0	0.1286	6.24E-01	1.37E+00	0.1147	2.02E+00	8.42E+00	0.1504
	T_3	**0**	0	0.1330	5.46E-01	1.31E+00	0.1223	5.49E+00	1.41E+01	0.1551
	T_4	**0**	0	0.1410	**4.46E-01**	9.65E-01	0.1283	**1.26E-01**	1.18E-01	0.1583
	Fonksiyon	f_4			f_5			f_6		
	Topoloji	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	-	4.26E-01	5.39E-01	5.3707	**0**	0	10.4910	3.79E-09	2.39E-09	4.8823
PABC	T_1	7.39E-13	1.75E-13	0.3787	**0**	0	0.7980	1.06E-10	2.08E-10	0.3563
	T_2	5.68E-13	1.10E-13	0.3977	**0**	0	0.8200	8.76E-11	1.90E-10	0.3653
	T_3	6.82E-13	1.11E-13	0.4157	**0**	0	0.8400	4.83E-11	8.44E-11	0.3770
	T_4	**6.25E-13**	1.05E-13	0.4273	**0**	0	0.8603	**1.83E-11**	2.73E-11	0.3817
	Fonksiyon	f_7			f_8			f_9		
	Topoloji	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	-	**0**	0	1.0220	2.90E-01	3.54E-02	9.2270	1.55E-11	6.82E-12	1.0073
PABC	T_1	**0**	0	0.1091	5.18E-02	3.31E-02	0.6530	2.37E-13	2.94E-13	0.1107
	T_2	**0**	0	0.1095	3.63E-02	2.32E-02	0.6713	4.10E-13	7.86E-13	0.1180
	T_3	**0**	0	0.1099	2.65E-02	2.21E-02	0.6883	1.13E-12	4.12E-12	0.1247
	T_4	**0**	0	0.1132	**2.36E-02**	2.31E-02	0.7040	**2.11E-13**	3.74E-13	0.1313
	Fonksiyon	f_{10}			f_{11}			f_{12}		
	Topoloji	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
ABC	-	**6.89E-04**	2.80E-04	1.0354	**0**	0	9.1827	**0**	0	8.2243
PABC	T_1	2.09E-03	1.94E-02	0.1032	**0**	0	0.7413	**0**	0	0.6357
	T_2	4.50E-03	2.45E-02	0.1113	**0**	0	0.7543	**0**	0	0.6543
	T_3	3.58E-03	7.30E-02	0.1154	**0**	0	0.7817	**0**	0	0.6823
	T_4	4.61E-02	1.43E-01	0.1177	**0**	0	0.8087	**0**	0	0.6967



Şekil 5.4. ABC ve farklı göç topolojileri ile gerçekleştirilen PABC algoritmalarının test fonksiyonları çözümleri için yakınsama grafikleri a)f_1, b)f_2, c)f_3, d)f_4, e)f_5, f)f_6, g)f_7 ve h)f_8



(a) f_9 (b) f_{10}

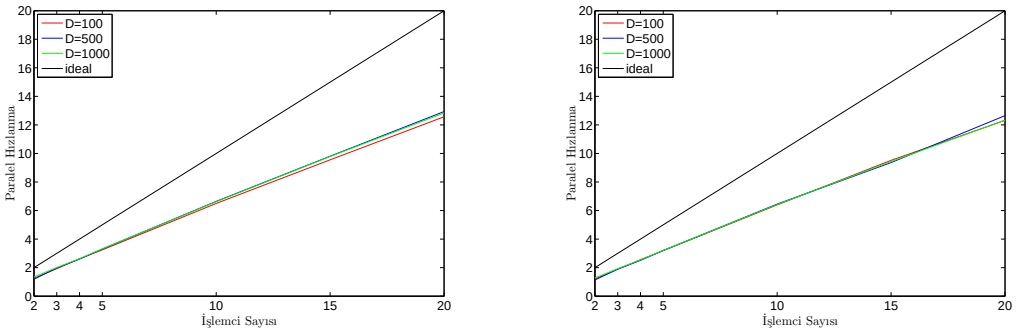
(c) f_{11} (d) f_{12}

Şekil 5.5. ABC ve farklı göç topolojileri ile gerçekleştirilen PABC algoritmalarının test fonksiyonları çözümleri için yakınsama grafikleri a)f_9, b)f_{10}, c)f_{11}, d)f_{12}

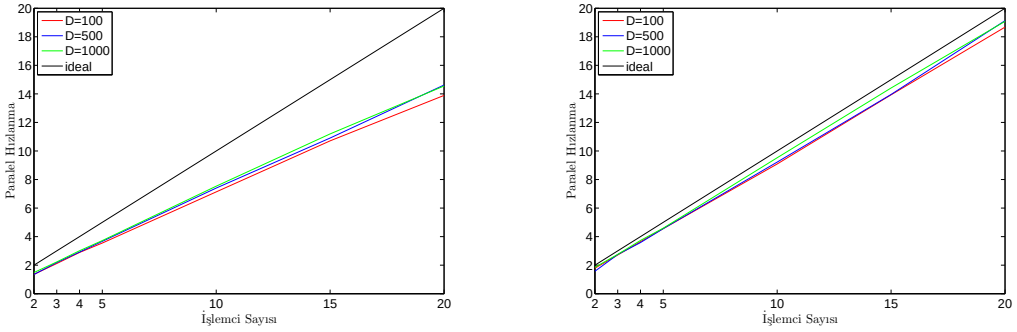
5.4.4. Popülasyon Büyüklüğü, Alt Popülasyon Sayısı, Göç Sıklığı ve Göç Topolojisinin Çalışma Zamanına Etkisi

Çalışmada incelenen bir diğer konu ise PABC algoritmasının çalışma zamanı bakımından analiz edilmesidir. Bu amaçla Tablo 5.7, Tablo 5.8 ve Tablo 5.9'daki sonuçlar hızlanma ve verimlilik analizleri için incelenmiştir. Şekil 5.6 ve Şekil 5.7'de ilgili tablolar için paralel hızlanma grafikleri verilmiştir.

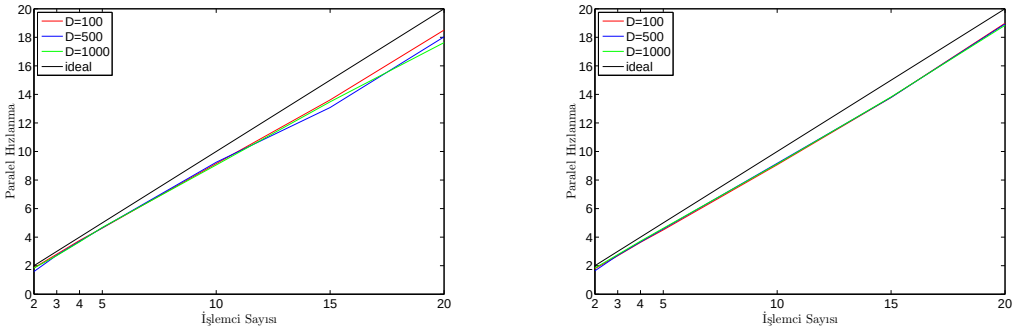
Çalışmada PABC algoritması ile yapılan tüm gerçekleştirimler kendi eşdeğerleri ile karşılaştırıldığında zamansal kazanç sağlamıştır. Bu çalışmalarda alt popülasyon sayısına eşit neredeyse doğrusala yakın paralel hızlanma elde edilmiştir. Ayrıca paralel verimliliğin, az hesaplama zamanı gerektiren test fonksiyonlarında daha az, yüksek hesaplama zamanı gerektiren test fonksiyonlarında daha fazla olduğu gözlemlenmiştir.



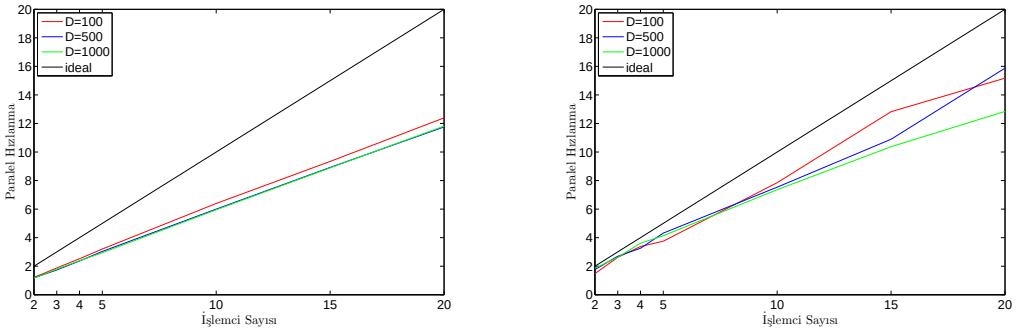
(a) f_1 (b) f_2



(c) f_3 (d) f_4

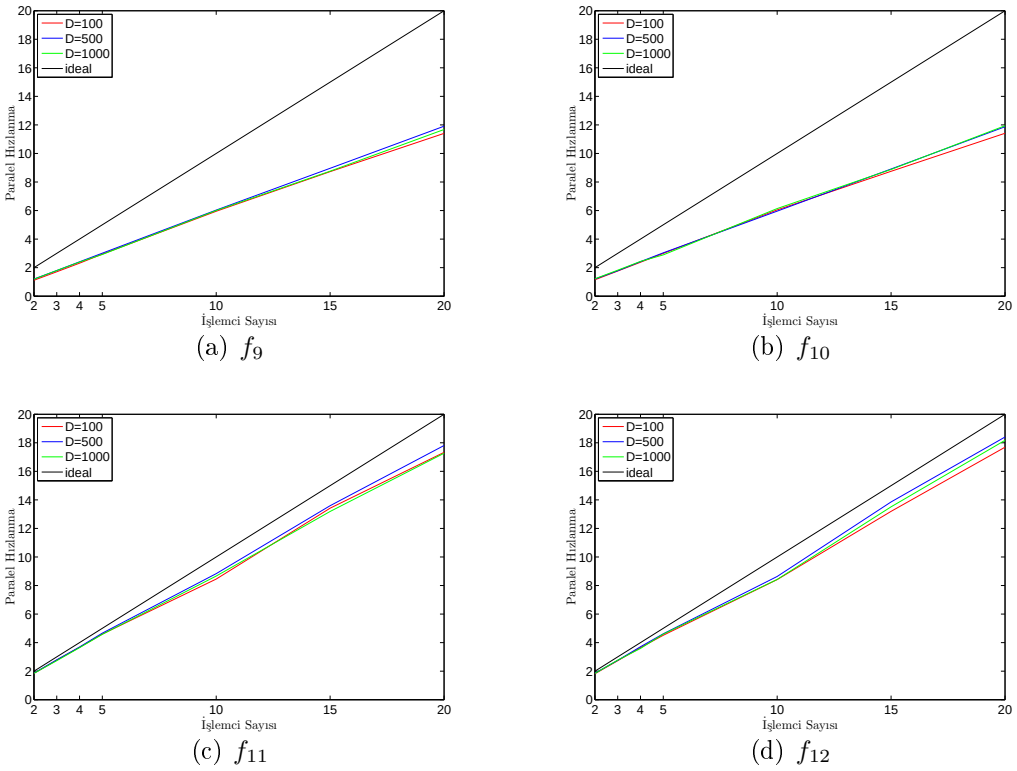


(e) f_5 (f) f_6



(g) f_7 (h) f_8

Şekil 5.6. PABC algoritması için paralel hızlanma grafikleri a)f_1, b)f_2, c)f_3, d)f_4, e)f_5, f)f_6, g)f_7 ve h)f_8



(a) f_9 (b) f_{10}

(c) f_{11} (d) f_{12}

Şekil 5.7. PABC algoritması için paralel hızlanma grafikleri a)f_9, b)f_{10}, c)f_{11}, d)f_{12}

Coarse-grained model ile paralel gerçekleştirimi yapılan sezgisel algoritmalarda alt popülasyonlar arası iletişim arttıkça iletişim maliyetinden kaynaklı algoritmanın çalışma zamanının da artması beklenir. Ancak bu çalışmada kullanılan paralel ağ mimarisi özellikleri sebebi ile çalışma zamanı önemli ölçüde etkilenmemiştir. Bu etki Tablo 5.15 ve Tablo 5.16'daki sırası ile f_4 ve f_6 fonksiyonları için göç sıklığı değerinin 10'dan 2500'e kadar ve farklı alt popülasyon sayıları (20, 15, 10, 5, 4 ve 3) için detaylıca görülebilir. Bu sonuçlardan gerçekleştirilen uygulamada göç sıklık değerlerinin PABC algoritmasının çalışma zamanı üzerinde büyük bir etki göstermediği gözlemlenmiştir.

Benzer şekilde Tablo 5.17'de verilen farklı topolojilerle gerçekleştirilen PABC modellerinin çalışma zamanları incelendiğinde; göç topolojisinin de PABC algoritmasının çalışma zamanı üzerinde büyük bir etkiye sahip olmadığı görülmüştür.

PABC algoritmasının çalışma zamanının göç topolojisine ve göç sıklığına duyarsızlığı, büyük oranda kullanılan yüksek başarımlı grid hesaplama merkezinin

özelliklerinden kaynaklanmaktadır. Faklı paralel hesaplama sistemleri üzerinde gerçekleştirilen uygulamalarda bu parametreler algoritmanın çalışma zamanını daha fazla etkileyebilir.

Genel olarak bu çalışmadan PABC algoritmasının alt popülasyon sayısı yüksek, göç sıklık aralığı düşük ve grid topolojisi kullanılarak büyük ölçekli problemler için başarılı bir şekilde çözüm üretilebileceği görülmüştür. Ayrıca çalışma zamanı bakımından alt popülasyon sayısının dolayısı ile işlemci sayısının çalışma süresi bakımından büyük önem taşıdığı, göç sıklığı ve göç topolojisinin ise çalışma süresini önemli ölçüde etkilemediği söylenebilir.

5.5. TLBO Algoritmasının Master-Slave, Coarse-Grained ve Hybrid Paralel Gerçekleştirimi

Eğitim ve öğretim sisteminden esinlenerek geliştirilen TLBO algoritmasından farklı tipteki optimizasyon problemlerine etkili çözümler sunması beklenmektedir. TLBO algoritmasının çok yeni olması sebebi ile literatürde sınırlı sayıda çalışma bulunmaktadır. Bu çalışmalar, TLBO algoritmasının test problemleri ve gerçek hayattaki uygulama problemleri çözümlerinde kullanılabilecek basit ve etkili tekniklerden biri olduğunu göstermektedir [165–168]. Çok amaçlı optimizasyon problemleri üzerine yapılan bazı çalışmalar da algoritmanın performansının diğer algoritmalar ile benzer olduğunu göstermiştir [169–171]. Ayrıca algoritmanın performansını artırmak için bazı yeni yaklaşımlar da önerilmiştir [28, 29]. Rao ve Patel temel TLBO algoritmasında bazı iyileştirmeler yaparak algoritmanın keşif yeteneğini arttırmışlar ve sınırlamasız test fonksiyonlarında başarılı sonuçlar almışlardır [28]. Başka bir çalışmalarında gelişmiş TLBO algoritmasının parametreleri olan elit birey sayısı, popülasyon büyüklüğü ve iterasyon sayısının performansa etkisini araştırmışlar, çeşitli sınırlamalı test fonksiyonları üzerinde keşif yeteneğini incelemişlerdir [29].

Popülasyon tabanlı sezgisel algoritmalarının etkin arama sürecini hızlandırmak için paralel modelleri üzerinde yapılan çalışmalar, TLBO algoritmasının çok yeni olması sebebi ile henüz yapılmamıştır. Dolayısı ile bu uygulamada diğer popülasyon

tabanlı algoritmaların paralelleştirilmesinde yaygın olarak kullanılan modellerin TLBO algoritmasına uygulanması, performans ve çalışma zamanlarının incelenmesi amaçlanmaktadır. Bu amaçla master-slave, coarse-grained ve hybrid paralelleştirme modelleri gerçekleştirilerek paralel TLBO (Parallel Teaching-Learning-Based Optimization, PTLBO) algoritmaları oluşturulmuştur.

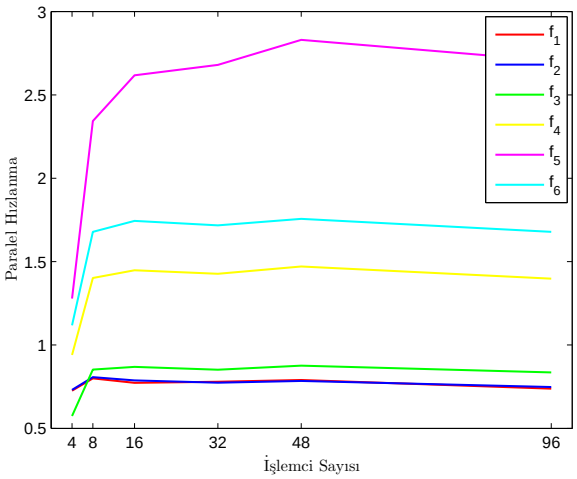
Yapılan bu uygulamada, gerçekleştirilen PTLBO algoritmalarının performans ve çalışma zamanı CEC2008 için özel olarak hazırlanmış büyük ölçekli ve zorlaştırılmış altı adet sürekli test fonksiyonu üzerinde incelenmiştir [159]. Tüm test fonksiyonlarının çözümü CEC2008 için tanımlanan problem boyutunda ve iterasyon sayısında gerçekleştirilmiştir.

5.5.1. Master-Slave PTLBO Algoritmasının İncelenmesi

Gerçekleştirilen master-slave paralelleştirme modelinde aday çözümlerinin uygunluk değerlerinin slave işlemciler tarafından eş zamanlı olarak hesaplanması sağlanırken, diğer algoritmik işlemler master işlemci tarafından gerçekleştirilmiştir. Nümerik test fonksiyonlarının çözümleri için geçen çalışma zamanı değerleri Tablo 5.18'de verilmiştir. Zaman kazançlarını yorumlayabilmek için paralel hızlanma ölçümleri dikkate alınmıştır. Bu ölçümlerin yapılması için 4, 8, 16, 32, 48 ve 96 adet işlemci kullanılmıştır. Master-slave PTLBO ($PTLBO_1$) algoritması için paralel hızlanma grafiği Şekil 5.8'de verilmiştir.

Tablo 5.18. TLBO ve farklı CPU sayıları ile koşulan $PTLBO_1$ algoritmalarının çalışma süreleri

Algoritma	CPU Sayısı	Fonksiyon					
		f_1	f_2	f_3	f_4	f_5	f_6
TLBO	1	7.86	7.92	8.88	14.87	28.84	17.83
$PTLBO_1$	4	10.83	10.84	15.46	15.81	22.56	15.95
	8	9.83	9.81	10.42	10.61	12.31	10.62
	16	10.17	10.06	10.22	10.27	11.02	10.22
	32	10.08	10.23	10.43	10.42	10.76	10.38
	48	9.95	10.10	10.14	10.11	10.19	10.15
	96	10.65	10.59	10.63	10.64	10.66	10.62



Şekil 5.8. $PTLBO_1$ algoritması için paralel hızlanma grafiği

Tablo 5.18 ve Şekil 5.8 incelendiğinde $PTLBO_1$ algoritması ile f_1, f_2 ve f_3 fonksiyonları için paralel hızlanma elde edilememiştir. Bunun yanında f_4, f_5 ve f_6 fonksiyonları için kısmi paralel hızlanma elde edilebilmiştir. Bu fonksiyonlarda elde edilen hızlanma değerleri fonksiyonların uygunluk değerlerinin hesaplanması için gerekli süre ile doğru orantılı olmuştur. Özellikle fonksiyon uygunluk değerinin hesaplanma süresinin yüksek olduğu f_5 fonksiyonunda en fazla hızlanma elde edilmiştir. Ayrıca, Şekil 5.8'deki grafikten de görüldüğü üzere paralel hızlanmanın belirli bir işlemci sayısına kadar arttığı, daha sonra duraksadığı da görülmektedir. Slave işlemci sayısının artması, bunların aynı anda master işlemci ile haberleşmek istemesi sebebi ile darboğaza neden olmaktadır. Genel olarak sonuçlar, master-slave paralelleştirme modelinin hızlanma ve verimliliği slave işlemcilerin işlem yükü ve master ve slave işlemciler arasındaki iletişim yükü arasındaki dengenin önemini göstermektedir.

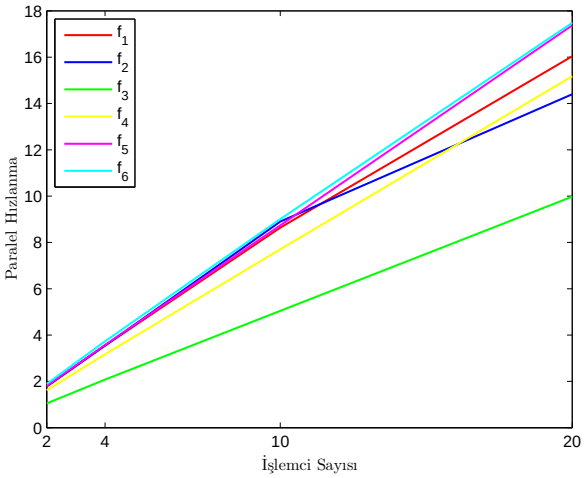
5.5.2. Coarse-Grained PTLBO Algoritmasının İncelenmesi

Gerçekleştirilen bir diğer uygulama coarse-grained PTLBO ($PTLBO_2$) algoritmasıdır. Bu uygulama için gerekli parametrelerden, göç topolojisi halka, göç sıklığı 20, göç eden birey sayısı 1 ve göç stratejisi olarak en iyi bireyin göç ederek en kötü bireyle yer değiştirdiği strateji benimsenmiştir. Alt popülasyon sayısının performansa ve hızlanmaya etkisini göstermek amacı ile de her biri farklı CPU'da

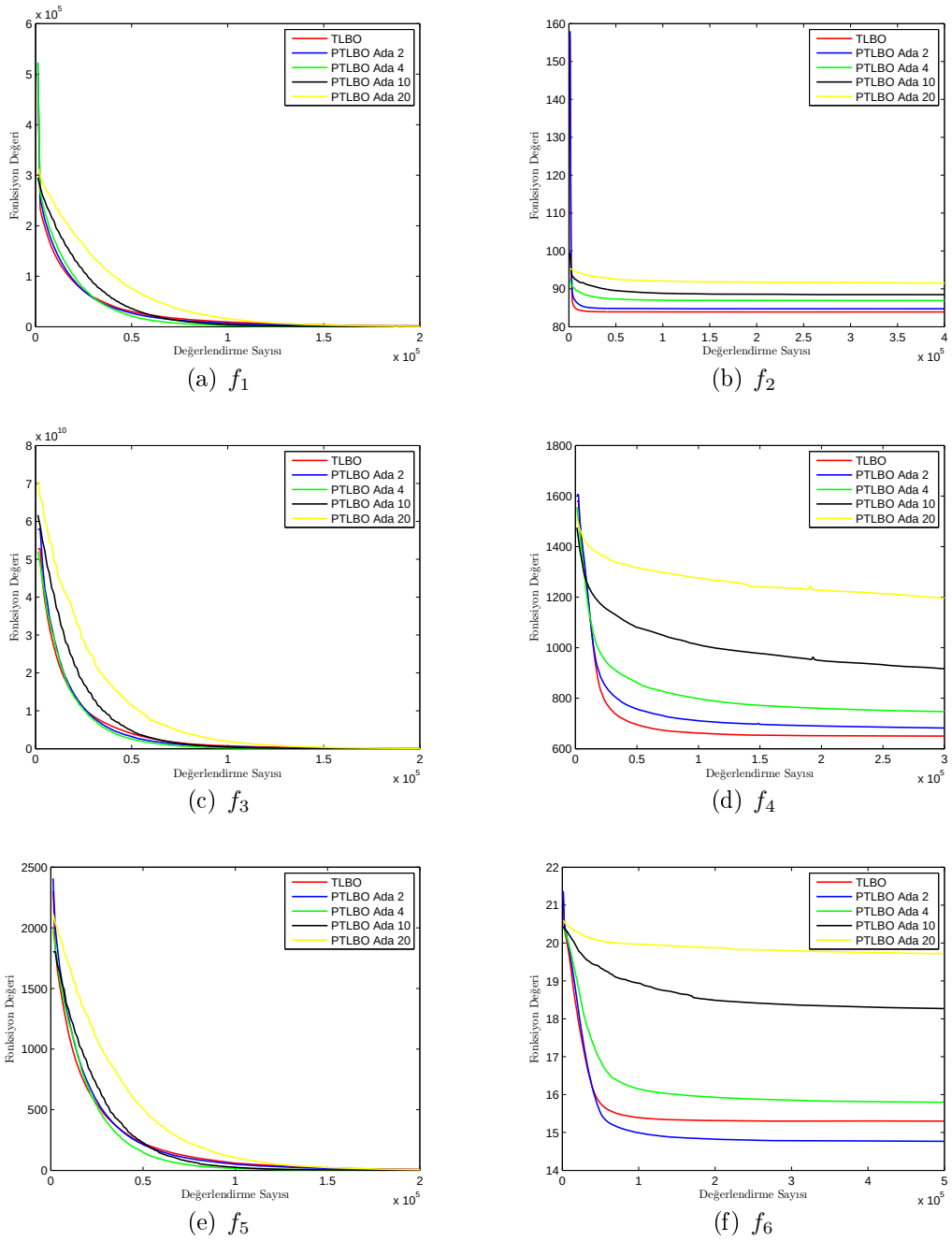
çalışan 2, 4, 10 ve 20 alt popülasyon incelenmiştir. Tablo 5.19'da nümerik test fonksiyonlarının çözümleri sonucu elde edilen ortalama uygunluk, standart sapma ve çalışma zamanı değerleri verilmiştir. Ayrıca Şekil 5.9'da $PTLBO_2$ algoritması için paralel hızlanma grafiği, Şekil 5.10'da standart seri TLBO ve 2, 4, 10 ve 20 alt popülasyondan oluşan $PTLBO_2$ algoritmalarının yakınsama grafikleri verilmiştir.

Tablo 5.19. TLBO ve farklı CPU sayıları ile koşulan $PTLBO_2$ algoritmalarının ortalama ve standart sapma değerleri

Algoritma	Fonksiyon	f_1			f_2		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman
TLBO	1	3.21E+00	8.02E+00	7.86	8.40E+01	2.59E+00	7.92
$PTLBO_2$	2	1.01E-02	1.58E-02	4.43	8.47E+01	9.93E-01	4.40
	4	4.20E-05	6.77E-05	2.22	8.51E+01	2.20E+00	2.23
	10	1.14E-04	3.58E-05	0.91	8.87E+01	2.15E+00	0.89
	20	6.43E-02	1.45E-02	0.49	9.13E+01	1.57E+00	0.55
Algoritma	Fonksiyon	f_3			f_4		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman
TLBO	1	8.73E+04	1.97E+05	8.88	6.41E+02	3.97E+01	14.87
$PTLBO_2$	2	4.23E+03	8.55E+03	8.47	6.45E+02	4.04E+01	9.22
	4	1.75E+03	1.75E+03	4.27	6.75E+02	2.63E+01	4.69
	10	2.14E+03	1.56E+03	1.76	8.15E+02	3.87E+01	1.93
	20	1.52E+04	2.54E+03	0.89	1.01E+03	3.93E+01	0.98
Algoritma	Fonksiyon	f_5			f_6		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman
TLBO	1	1.33E+00	2.93E+00	28.84	1.49E+01	1.10E+00	17.83
$PTLBO_2$	2	5.12E-01	6.69E-01	15.93	1.51E+01	1.04E+00	9.44
	4	3.62E-01	1.26E+00	8.07	1.53E+01	2.07E+00	4.78
	10	1.75E-03	2.81E-03	3.30	1.66E+01	3.11E+00	1.98
	20	4.03E-02	2.11E-02	1.66	1.94E+01	1.92E-01	1.02



Şekil 5.9. $PTLBO_2$ algoritması için paralel hızlanma grafiği



(a) f_1 (b) f_2

(c) f_3 (d) f_4

(e) f_5 (f) f_6

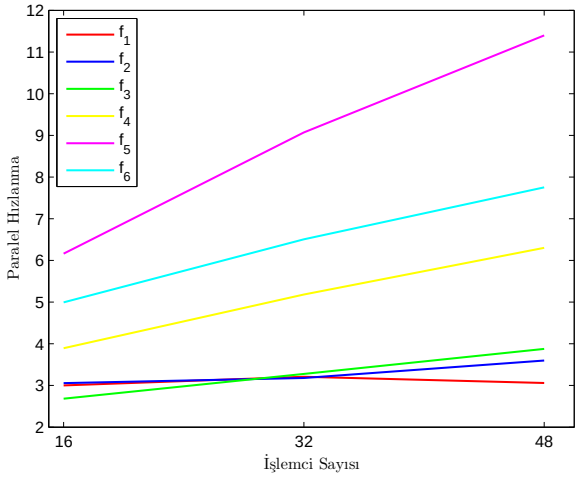
Şekil 5.10. TLBO ve farklı CPU sayıları ile gerçekleştirilen $PTLBO_2$ algoritmalarının test fonksiyonları çözümleri için yakınsama grafikleri a)f_1, b)f_2, c)f_3, d)f_4, e)f_5 ve f)f_6

Tablo 5.19'daki sonuçlar incelendiğinde $PTLBO_2$ algoritması ile f_2, f_4 ve f_6 fonksiyonlarının çözüm kalitesinde herhangi bir değişiklik olmamıştır. Bunun yanında f_1, f_3 ve f_5 fonksiyonlarında $PTLBO_2$ algoritması ile çözüm kalitelerinde iyileşme gözlemlenmiştir. Alt popülasyon sayısının değişimi incelendiğinde, algoritmada popülasyonun daha küçük parçalara bölünmesi genel olarak

algoritmanın performansını değiştirmezken çözüm süresini kısaltmıştır. Şekil 5.9'da verilen paralel hızlanma grafiği incelendiğinde çalışmada hızlanmanın kullanılan işlemci sayısına yakın olduğu doğrusal hızlanma elde edilmiştir. Bununla beraber Şekil 5.10'de verilen yakınsama grafikleri incelendiğinde ise seri ve paralel modellerin yakınsama hızları birbirlerine yakın çıkmıştır.

5.5.3. Hybrid PTLBO Algoritmasının İncelenmesi

Gerçekleştirilen bir diğer uygulama hybrid PTLBO ($PTLBO_3$) algoritmasıdır. Coarse-grained ve master-slave modellerinin avantajlarının birleştirildiği bu hybrid model iki seviyeden oluşmaktadır. Üst seviyede coarse-grained modeli, alt seviyede master-slave modeli bulunmaktadır. Bu uygulama için gerekli parametrelerden, alt popülasyon sayısı 4, göç topolojisi halka, göç sıklığı 20, göç eden birey sayısı 1 ve göç stratejisi olarak en iyi bireyin göç ederek en kötü bireyle yer değiştirdiği strateji benimsenmiştir. Slave işlemci sayısının performansa ve hızlanmaya etkisini göstermek amacı ile de her biri farklı CPU da çalışan 4 alt popülasyon ile 12, 28 ve 44 slave işlemcilerin eşit sayılarda dağıtıldığı yapılar ayrı ayrı analiz edilmiştir. Tablo 5.20'de nümerik test fonksiyonlarının çözümleri sonucu elde edilen ortalama uygunluk, standart sapma ve çalışma zamanı değerleri verilmiştir. Ayrıca, $PTLBO_3$ algoritması için paralel hızlanma grafiği Şekil 5.11'de verilmiştir.



Şekil 5.11. $PTLBO_3$ algoritması için paralel hızlanma grafiği

Tablo 5.20. TLBO ve farklı CPU sayıları ile koşulan $PTLBO_3$ algoritmalarının ortalama ve standart sapma değerleri

Algoritma	Fonksiyon	f_1			f_2		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman
TLBO	1	3.21E+00	8.02E+00	7.86	8.40E+01	2.59E+00	7.92
$PTLBO_3$	4+12	4.56E-06	2.80E-05	2.62	8.30E+01	8.72E+01	2.59
	4+28	5.08E-06	1.23E-05	2.45	8.38E+01	8.83E+01	2.49
	4+44	6.03E-06	5.43E-05	2.57	8.59E+01	9.01E+01	2.20
Algoritma	Fonksiyon	f_3			f_4		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman
TLBO	1	8.73E+04	1.97E+05	8.88	6.41E+02	3.97E+01	14.87
$PTLBO_3$	4+12	3.31E+03	2.10E+03	3.31	6.56E+02	6.98E+02	3.82
	4+28	1.25E+03	5.56E+03	2.71	6.63E+02	7.35E+02	2.87
	4+44	1.47E+03	8.59E+03	2.29	6.97E+02	7.71E+02	2.36
Algoritma	Fonksiyon	f_5			f_6		
	CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman
TLBO	1	1.33E+00	2.93E+00	28.84	1.49E+01	1.10E+00	17.83
$PTLBO_3$	4+12	3.72E-02	2.08E-01	4.68	1.35E+01	1.73E+01	3.57
	4+28	1.23E-01	7.77E-01	3.18	1.43E+01	1.76E+01	2.74
	4+44	1.95E-01	1.73E+00	2.53	1.61E+01	1.83E+01	2.30

Tablo 5.20'deki sonuçlar incelendiğinde $PTLBO_3$ algoritması ile f_1, f_3 ve f_5 fonksiyonlarının çözüm kalitesi iyileşirken diğerleri değişmemiştir. Ayrıca uygunluk değerinin hesaplanma süresi yüksek olan test fonksiyonlarında daha fazla hızlanma elde edilmiştir. Paralel hızlanma grafiği incelendiğinde işlemci sayısı arttıkça verimliliğinde düştüğü görülmektedir. Diğer taraftan $PTLBO_3$ algoritması ile aynı sayıda alt popülasyon içeren $PTLBO_2$ algoritması karşılaştırılacak olursa, çözüm kalitesi açısından benzer performans gösterdiği söylenebilir. Bunun yanında $PTLBO_2$ algoritma modelinde işlemciler arası iletişim daha az olduğu için hızlanmanın daha fazla olduğu görülmektedir.

Yeni bir sezgisel algoritma olan TLBO ile yapılan bu çalışmalarda algoritmanın karakteristiği bazı paralel modeller üzerinde analiz edilmeye çalışılmıştır. Özetle sonuçlardan coarse-grained paralelleştirme modelinin düşük çalışma zamanı ile çözüm kalitesini koruduğu veya iyileştirdiği, master-slave ve hybrid paralelleştirme modellerinin düşük maliyetli test problemlerinde sınırlı hızlanma sağladığı görülmüştür.

5.6. ABC, CS, DE, FF, GS, PSO ve TLBO Algoritmalarının Seri ve Paralel Modellerinin Kıyaslanması

Bu uygulamada popülasyon tabanlı sezgisel algoritmalardan ABC, CS, DE, FF, GS, PSO ve TLBO algoritmaları incelenmiştir. Bu algoritmaların seri modelleri ile ilgili oldukça fazla sayıda çalışma bulunmasına rağmen paralel modelleri ile yapılan çalışmalar sınırlıdır. Ayrıca, GS ve TLBO algoritmalarının paralel modelleri ile ilgili çalışmalar literatürde henüz bulunmamaktadır. Bunun yanı sıra yapılan literatür taramasında populasyon tabanlı sezgisel algoritmaların paralel modellerinin birbirleri ile karşılaştırıldığı bir çalışmaya da rastlanamamıştır.

Bu uygulamada gerçekleştirilen algoritmaların seri ve coarse-grained paralel modellerinin kıyaslanması amaçlanmaktadır. Seri modelleri karşılaştırılırken popülasyon tabanlı sezgisel algoritmaların ortak parametresi olan popülasyon büyüklüğünün çözüm kalitesine ve çalışma zamanına etkisi her bir algoritma için detaylıca incelenmiştir. Paralel modelleri karşılaştırılırken ise işlemci sayısının çözüm kalitesine ve çalışma zamanına etkisi detaylıca incelenmiştir.

Çalışmada ABC, CS, DE, FF, GS, PSO, ve TLBO algoritmalarının coarse-grained paralel gerçekleştirimleri sırası ile PABC, PCS, PDE, PFF, PGS, PPSO, ve PTLBO olarak gösterilmiştir. Bilindiği üzere coarse-grained paralel gerçekleştirim algoritmalara ek yeni kontrol parametreler getirmektedir. Bu parametre değerleri algoritmaların çalışma zamanını ve performansını kısmen etkilemelerine rağmen, bu çalışmada her bir algoritma için en iyi değerler araştırılmamıştır. Bunun yerine tüm algoritmalar için iyi bilinen ve kolayca uyarlanabilecek aynı değerler seçilmiştir.

Yapılan bu çalışmada da, CEC2008 için özel olarak hazırlanmış zorlaştırılmış altı adet test fonksiyonu ile yaygın olarak kullanılan altı adet test fonksiyonu kullanılmıştır. Test fonksiyonları için problem boyutları 100, iterasyon sayısı 500000 alınmıştır. Bu değerler CEC2008 yarışması için tanımlanan özel değerlerdir. Algoritmaların seri modellerinde popülasyon büyüklüğünün performansa ve çalışma zamanına etkisinin incelenmesi amacıyla popülasyon büyüklüğü 96, 48, 24 ve 12 değerleri için incelenmiştir. Algoritmaların paralel modellerinde işlemci sayısının performansa ve çalışma zamanına etkisinin incelenmesi amacıyla işlemci sayısı 2,

4, 8 ve 16 değerleri için incelenmiştir. Coarse-grained paralelleştirme modeli için gerekli diğer parametreler, göç topolojisi halka, göç sıklığı 10, göç eden birey sayısı 1 ve göç stratejisi olarak en iyi bireyin göç ederek en kötü bireyle yer değiştirdiği strateji seçilmiştir. Ayrıca algoritmalara özel diğer parametre değerleri: ABC için limit 1000, CS için adım büyüklüğü ve mutasyon oranı sırası ile 1.50 ve 0.25, DE için ölçekleme faktörü ve çaprazlama oranı sırası ile 0.5 ve 0.3, FF için α, β_o ve γ sırası ile 0.25, 0.20 ve 1, GS için G_0 ve α sırası ile 100 ve 20, PSO için öğrenme faktörleri ve hız vektörü ağırlık çarpanı sırası ile 1.8 ve 0.6 olarak ayarlanmıştır. TLBO algoritması ise herhangi bir özel kontrol parametresi içermemektedir.

5.6.1. Popülasyon Büyüklüğünün Çözüm Kalitesine ve Çalışma Zamanına Etkisi

ABC, CS, DE, FF, GS, PSO, ve TLBO algoritmalarının performans sonuçları sırası ile Tablo 5.21 - 5.27'de verilmiştir. Anlamlı bir karşılaştırma yapmak için Tablo 5.28 ve Tablo 5.29 sunulmuştur. Tablo 5.28'de algoritmalar her bir test fonksiyonu için göstermiş oldukları performansa göre sıralanmaktadır. Tablo 5.29'da ise Tablo 5.28'deki sıralama değerlerinin toplamı gösterilmekte ve algoritmaların genel sıralaması verilmektedir. Algoritmaların çalışma zamanları Tablo 5.30'da verilmekte ve bu tablonun grafiksel gösterimi Şekil 5.12'de gösterilmektedir. Ayrıca algoritmaların hesaplama karmaşıklığı CEC2006'da verilen yöntemle hesaplanmıştır [172]. Bu yöntemde bir algoritmanın karmaşıklığı $(T_2 - T_1)/T_1$ formülü ile hesaplanır. Burada T_1 10,000 fonksiyon hesaplanması için geçen ortalama zaman, T_2 ise algoritmanın 10,000 fonksiyon değerlendirmesi yaptığında geçen ortalama zamandır. Tüm test fonksiyonları dikkate alındığında hesaplama zamanı $T_1 = 0.0733$ çıkmaktadır. Bu değer üzerinden hesaplanan algoritmaların hesaplama karmaşıklıkları Tablo 5.31'de verilmektedir. Ayrıca, Şekil 5.13'de tüm algoritmaların tüm test fonksiyonları çözümleri için geçen sürelerin ortalama değerleri grafiksel olarak gösterilmiştir.

Popülasyon büyüklüğünün algoritmaların çözüm kalitesine etkisi Tablo 5.21 - 5.27'dan görüldüğü üzere algoritmadan algoritmaya farklılık göstermektedir. DE gibi bazı algoritmalarda en iyi sonuçlar yüksek popülasyon büyüklüğünde elde edilirken, ABC gibi bazı algoritmalarda en iyi sonuçlar düşük popülasyon

büyüklüğünde elde edilmektedir. Tablo 5.28 ve Tablo 5.29'da ortalama çözüm kalitelerine dayalı olarak algoritmaların sıralanmasını göstermektedir. Tablo 5.28'deki sonuçlar ABC algoritmasının tüm popülasyon büyüklükleri için f_3, f_6, f_7, f_{10}, f_{11} ve f_{12} fonksiyonlarında diğer tüm algoritmalardan daha üstün olduğu görülmektedir. f_1, f_4, f_5 ve f_9 fonksiyonları için ABC algoritması düşük popülasyon büyüklüklerinde, DE algoritması yüksek popülasyon büyüklüklerinde diğerlerinden daha iyi sonuçlar üretmektedir. Diğer taraftan f_2 fonksiyonu için FF algoritması ve f_8 fonksiyonu için TLBO algoritması diğerlerine göre daha kaliteli çözümler üretmiştir.

Tablo 5.30 ve Şekil 5.12'de tüm algoritmalar çalışma zamanı açısından karşılaştırıldıklarında, popülasyon büyüdükçe FF ve GS algoritmalarının çalışma zamanı önemli derecede artmakta, DE ve PSO algoritmalarının çalışma zamanı çok az azalmakta, ABC, CS ve TLBO algoritmalarının çalışma zamanı ise kayda değer bir oranda değişmemektedir. Çalışma zamanlarının özetlendiği Şekil 5.13 incelendiğinde en hızlı algoritmanın ABC, en yavaş algoritmanın GS olduğu görülmektedir. Tüm fonksiyonların ortalama çalışma süreleri kıyaslandığında ABC algoritması GS algoritmasından popülasyon büyüklüğü 12 olduğunda 12.45 kat, popülasyon büyüklüğü 96 olduğunda 82.93 kat daha hızlı çalışmıştır. Sonuçlardan FF ve GS algoritmalarının çok yavaş olduğu görülmektedir.

Tablo 5.21. Farklı popülasyon büyüklükleri için seri ABC deneysel sonuçları

Fonksiyon	Popülasyon Büyüklüğü							
	12		24		48		96	
	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.
f_1	3.98E-13	1.88E-13	3.98E-13	1.64E-13	3.98E-13	1.68E-13	3.41E-13	2.25E-13
f_2	1.92E+01	6.04E+00	3.04E+01	6.49E+00	5.34E+01	6.16E+00	7.69E+01	5.27E+00
f_3	4.21E+01	4.91E+01	3.74E+01	3.67E+01	1.13E+01	1.90E+01	8.62E+00	1.07E+01
f_4	1.71E-13	1.80E-13	1.14E-13	1.98E-13	1.71E-13	1.39E-13	2.55E-01	4.21E-01
f_5	7.31E-11	3.47E-10	2.65E-10	1.42E-09	3.13E-13	1.73E-13	3.13E-13	3.52E-14
f_6	1.31E-12	1.18E-13	1.22E-12	7.84E-14	1.19E-12	7.85E-14	3.03E-09	2.62E-09
f_7	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
f_8	2.47E-01	4.11E-02	2.31E-01	3.48E-02	2.46E-01	4.00E-02	3.10E-01	3.21E-02
f_9	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	1.97E-11	1.22E-11
f_{10}	2.12E-01	1.12E+00	4.20E-08	2.08E-07	5.71E-07	8.60E-07	7.16E-04	3.88E-04
f_{11}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
f_{12}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00

Tablo 5.22. Farklı popülasyon büyüklükleri için seri CS deneysel sonuçları

Fonksiyon	Popülasyon Büyüklüğü							
	12		24		48		96	
	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.
f_1	4.00E+02	2.11E+03	4.55E-12	8.72E-12	1.14E-13	2.61E-13	2.95E-07	1.19E-07
f_2	1.21E+02	9.89E+00	9.89E+01	7.07E+00	7.25E+01	7.64E+00	4.81E+01	5.51E+00
f_3	2.78E+06	1.18E+07	1.73E+02	5.80E+01	2.19E+02	7.24E+01	3.10E+02	1.50E+02
f_4	3.96E+02	6.41E+01	4.02E+02	4.30E+01	4.70E+02	2.87E+01	5.32E+02	4.31E+01
f_5	3.66E+00	1.23E+01	3.69E-02	4.49E-02	9.86E-04	3.02E-03	6.15E-05	1.65E-04
f_6	2.02E+01	3.72E-01	2.01E+01	6.11E-01	1.89E+01	4.61E+00	2.05E+01	3.40E-01
f_7	2.34E+04	7.71E+03	6.02E+02	7.62E+02	3.47E+00	7.50E+00	3.33E-02	1.80E-01
f_8	1.39E+01	1.06E+01	3.91E-01	1.86E-01	1.50E-01	2.50E-02	1.40E-01	2.44E-02
f_9	4.92E-01	1.48E+00	0.00E+00	0.00E+00	1.09E-14	8.11E-15	1.76E-04	3.09E-05
f_{10}	1.48E+03	5.15E+03	1.36E+01	7.54E+00	5.24E+00	6.03E+00	2.45E+00	3.88E+00
f_{11}	1.92E+03	9.47E+03	2.12E+00	2.64E+00	2.99E+00	7.02E-01	1.85E+00	3.75E-01
f_{12}	2.48E+06	9.95E+06	5.81E+01	2.40E+01	4.26E+01	2.00E+01	1.16E+01	1.58E+01

Tablo 5.23. Farklı popülasyon büyüklükleri için seri DE deneysel sonuçları

Fonksiyon	Popülasyon Büyüklüğü							
	12		24		48		96	
	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.
f_1	1.74E+01	6.91E+01	5.33E-01	1.43E+00	2.74E-02	1.04E-01	5.68E-14	1.16E-13
f_2	9.97E+01	1.47E+01	1.97E+00	4.66E+00	6.65E+00	4.23E-01	2.50E+01	1.58E+00
f_3	2.10E+05	1.13E+06	1.36E+02	3.32E+01	1.55E+02	2.10E+01	1.96E+02	2.71E+01
f_4	8.85E+00	3.75E+00	6.20E-01	8.16E-01	1.33E-01	6.04E-01	1.71E-13	1.13E-13
f_5	3.21E-01	3.96E-01	1.67E-01	3.27E-01	2.84E-14	2.84E-14	2.84E-14	9.68E-14
f_6	5.41E-01	6.59E-01	3.79E-02	2.04E-01	1.14E-13	5.80E-14	3.67E-09	2.96E-10
f_7	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
f_8	2.80E-02	4.28E-03	4.95E-02	5.11E-03	8.95E-02	9.47E-03	1.74E-01	2.00E-02
f_9	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	2.07E-09	1.39E-10
f_{10}	5.54E+00	4.83E+00	1.20E+00	2.20E+00	5.13E-01	3.71E-02	6.03E-01	8.71E-02
f_{11}	3.97E-04	2.05E-03	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
f_{12}	7.19E-09	3.87E-08	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00

Tablo 5.24. Farklı popülasyon büyüklükleri için seri FF deneysel sonuçları

Fonksiyon	Popülasyon Büyüklüğü							
	12		24		48		96	
	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.
f_1	1.32E-03	1.25E-04	1.75E-03	1.51E-04	2.75E-03	3.41E-04	4.87E-03	5.54E-04
f_2	1.40E-01	2.75E-02	1.28E-01	1.77E-02	1.18E-01	1.58E-02	9.98E-02	1.64E-02
f_3	2.99E+03	4.57E+03	2.76E+03	3.95E+03	2.69E+03	4.48E+03	3.17E+03	4.09E+03
f_4	2.72E+02	4.57E+01	2.06E+02	3.05E+01	1.70E+02	3.53E+01	1.46E+02	2.43E+01
f_5	2.58E-03	3.57E-03	2.25E-03	1.42E-03	2.68E-03	2.85E-04	3.22E-03	5.39E-04
f_6	4.69E-03	2.44E-04	5.50E-03	3.12E-04	6.55E-03	3.99E-04	9.06E-03	6.53E-04
f_7	8.67E-01	1.73E+00	6.67E-02	2.49E-01	0.00E+00	0.00E+00	0.00E+00	0.00E+00
f_8	9.81E-02	1.57E-02	2.98E-02	5.37E-03	9.27E-03	2.04E-03	4.55E-03	1.03E-03
f_9	6.12E-02	4.02E-02	1.05E-01	6.53E-02	1.49E-01	5.95E-02	2.73E-01	8.11E-02
f_{10}	6.15E-01	1.64E-01	1.12E+00	7.46E-01	1.58E+00	1.83E+00	6.22E+00	6.07E+00
f_{11}	9.86E-07	1.29E-07	1.04E-03	5.58E-03	2.09E-06	2.68E-07	4.53E-06	7.99E-07
f_{12}	1.88E-03	4.10E-03	8.01E-04	2.74E-03	1.06E-04	1.24E-05	2.32E-04	3.84E-05

Tablo 5.25. Farklı popülasyon büyüklükleri için seri GS deneysel sonuçları

Fonksiyon	Popülasyon Büyüklüğü							
	12		24		48		96	
	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.
f_1	3.60E+02	1.95E+02	1.91E+00	7.13E+00	0.00E+00	3.28E-14	9.36E+00	3.78E+01
f_2	4.80E+01	8.05E+00	5.53E+01	4.05E+00	6.07E+01	2.13E+00	6.82E+01	1.99E+00
f_3	1.13E+07	1.73E+07	3.15E+02	3.21E+02	1.72E+03	1.59E+03	4.32E+05	3.43E+05
f_4	1.56E+03	7.52E+01	5.99E+02	5.97E+01	8.45E+01	1.80E+01	5.07E+01	5.91E+00
f_5	1.19E+00	1.82E+00	2.39E+00	4.27E+00	7.56E+02	5.32E+01	1.77E+03	6.65E+01
f_6	1.01E+01	2.38E+00	1.87E-08	2.00E-09	1.17E-08	7.60E-10	8.50E-09	5.93E-10
f_7	6.67E-02	2.49E-01	1.33E-01	7.18E-01	0.00E+00	0.00E+00	0.00E+00	0.00E+00
f_8	2.07E+03	2.36E+02	4.15E+02	1.82E+02	9.45E-02	3.77E-02	3.42E-02	1.42E-02
f_9	1.84E+02	1.08E+01	2.67E-07	1.89E-08	1.72E-07	1.05E-08	1.31E-07	6.43E-09
f_{10}	1.43E+06	3.01E+05	5.00E-01	2.81E-05	5.05E-01	1.46E-02	5.31E-01	5.59E-02
f_{11}	8.29E-03	1.59E-02	1.55E-02	2.75E-02	5.18E-03	1.41E-02	2.07E-03	7.76E-03
f_{12}	7.36E-02	3.62E-01	7.40E-02	2.67E-01	1.36E-03	3.51E-03	1.71E-03	3.87E-03

Tablo 5.26. Farklı popülasyon büyüklükleri için seri PSO deneysel sonuçları

Fonksiyon	Popülasyon Büyüklüğü							
	12		24		48		96	
	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.
f_1	1.77E+04	2.36E+04	8.78E+03	7.38E+03	8.44E+03	4.51E+03	6.88E+03	3.94E+03
f_2	7.10E+01	1.23E+01	5.37E+01	1.10E+01	3.56E+01	1.45E+01	2.77E+01	1.58E+01
f_3	9.24E+08	1.69E+09	3.28E+08	7.45E+08	4.30E+08	6.90E+08	2.20E+08	3.74E+08
f_4	9.41E+02	8.28E+01	8.23E+02	7.61E+01	7.74E+02	8.27E+01	7.46E+02	7.58E+01
f_5	8.14E+01	5.93E+01	7.13E+01	5.02E+01	5.34E+01	3.20E+01	5.23E+01	3.80E+01
f_6	1.93E+01	1.92E-01	1.88E+01	2.03E+00	1.68E+01	4.97E+00	1.09E+01	6.76E+00
f_7	1.66E+03	7.23E+02	2.05E+02	1.91E+02	4.74E+01	4.90E+01	1.92E+01	4.54E+01
f_8	4.70E-01	9.17E-01	3.98E-02	2.45E-02	2.71E-02	7.75E-03	2.70E-02	6.00E-03
f_9	1.13E+01	5.64E+00	3.33E-01	1.80E+00	1.17E-11	5.42E-11	9.34E-07	5.01E-06
f_{10}	1.28E+03	6.52E+03	1.27E+01	4.23E+01	1.69E+00	3.67E+00	4.33E+00	6.71E+00
f_{11}	7.43E-01	6.29E-01	1.81E-01	3.25E-01	1.83E-01	3.68E-01	1.52E-01	2.39E-01
f_{12}	1.08E+01	1.60E+01	7.45E-01	1.08E+00	4.83E-01	1.02E+00	9.27E-02	4.47E-01

Tablo 5.27. Farklı popülasyon büyüklükleri için seri TLBO deneysel sonuçları

Fonksiyon	Popülasyon Büyüklüğü							
	12		24		48		96	
	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.
f_1	1.23E+05	2.09E+04	1.28E+04	7.27E+03	1.02E+03	1.58E+03	9.10E+02	1.70E+03
f_2	9.69E+01	1.27E+00	9.55E+01	1.59E+00	9.25E+01	2.04E+00	8.86E+01	2.31E+00
f_3	3.04E+10	9.90E+09	2.59E+09	1.49E+09	1.38E+08	2.32E+08	2.54E+07	7.03E+07
f_4	1.16E+03	7.29E+01	8.65E+02	3.87E+01	7.77E+02	5.03E+01	7.11E+02	4.03E+01
f_5	1.05E+03	1.54E+02	1.32E+02	7.00E+01	1.98E+01	1.95E+01	8.95E+00	1.11E+01
f_6	1.98E+01	1.35E-01	1.92E+01	1.48E-01	1.88E+01	2.22E-01	1.79E+01	4.88E-01
f_7	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
f_8	2.34E-05	1.21E-05	4.64E-05	2.08E-05	9.19E-05	3.72E-05	1.78E-04	4.33E-05
f_9	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
f_{10}	9.81E-01	2.52E-02	5.00E-01	4.75E-12	5.00E-01	1.61E-15	5.00E-01	1.58E-16
f_{11}	2.43E-01	6.70E-02	1.87E-02	1.92E-02	1.25E-03	1.43E-03	5.83E-04	8.13E-04
f_{12}	9.43E+00	4.90E-01	7.45E+00	1.61E+00	3.13E+00	1.85E+00	1.06E+00	5.23E-01

Tablo 5.28. ABC, CS, DE, FF, GS, PSO ve TLBO algoritmalarının performans sıralaması

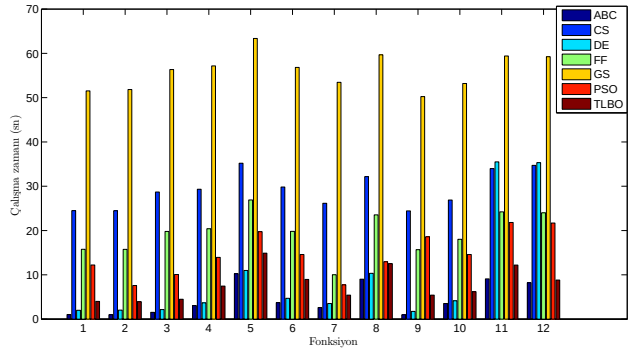
NP	Algoritma	Fonksiyon											
		f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}
12	ABC	1	2	1	1	1	1	1	4	1	1	1	1
	CS	5	7	4	4	5	7	5	6	3	6	7	7
	DE	3	6	3	2	3	3	1	2	1	4	3	2
	FF	2	1	2	3	2	2	3	3	2	2	2	3
	GS	4	3	5	7	4	4	2	7	5	7	4	4
	PSO	6	4	6	5	6	5	4	5	4	5	6	6
	TLBO	7	5	7	6	7	6	1	1	1	3	5	5
24	ABC	1	3	1	1	1	1	1	5	1	1	1	1
	CS	2	7	3	4	3	7	5	6	1	6	6	6
	DE	4	2	2	2	4	4	1	4	1	4	1	1
	FF	3	1	5	3	2	3	2	2	3	3	2	2
	GS	5	5	4	5	5	2	3	7	2	2	3	3
	PSO	6	4	6	6	6	5	4	3	4	5	5	4
	TLBO	7	6	7	7	7	6	1	1	1	2	4	5
48	ABC	3	4	1	1	2	2	1	7	1	1	1	1
	CS	2	6	3	5	3	7	2	6	2	7	6	6
	DE	5	2	2	2	1	1	1	4	1	4	1	1
	FF	4	1	5	4	4	4	1	2	5	5	2	2
	GS	1	5	4	3	7	3	1	5	4	3	4	3
	PSO	7	3	7	6	6	5	3	3	3	6	5	4
	TLBO	6	7	6	7	5	6	1	1	1	2	3	5
96	ABC	2	6	1	2	2	1	1	7	2	1	1	1
	CS	3	4	3	5	3	7	2	5	6	5	6	6
	DE	1	2	2	1	1	2	1	6	3	4	1	1
	FF	4	1	4	4	4	4	1	2	7	7	2	2
	GS	5	5	5	3	7	3	1	4	4	3	4	3
	PSO	7	3	7	7	6	5	3	3	5	6	5	4
	TLBO	6	7	6	6	5	6	1	1	1	2	3	5

Tablo 5.29. ABC, CS, DE, FF, GS, PSO ve TLBO algoritmalarının performans sıralama değerleri toplamı ve genel performans sıralamaları

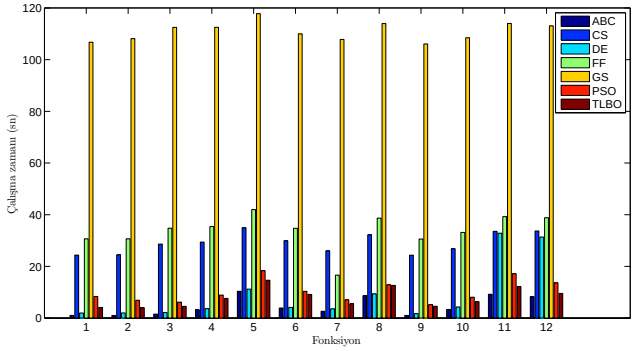
NP		Algoritma						
		ABC	CS	DE	FF	GS	PSO	TLBO
12	Toplam	16	66	33	27	56	62	54
	Genel Performans Sırası	1	7	3	2	5	6	4
24	Toplam	18	56	30	31	46	58	54
	Genel Performans Sırası	1	6	2	3	4	7	5
48	Toplam	25	55	25	39	43	58	50
	Genel Performans Sırası	1	5	1	2	3	6	4
96	Toplam	27	55	25	42	47	61	49
	Genel Performans Sırası	2	6	1	3	4	7	5

Tablo 5.30. Farklı popülasyon büyüklükleri ile koşulan ABC, CS, DE, FF, GS, PSO ve TLBO algoritmalarının çalışma süreleri

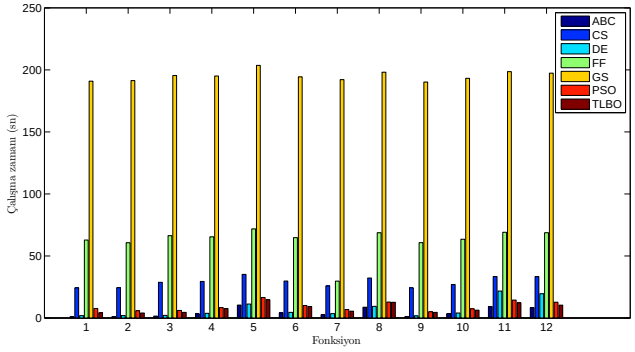
Algoritma	NP	Fonksiyon												Avg.
		f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	
ABC	12	1.02	0.99	1.54	3.06	10.25	3.71	2.60	9.02	0.98	3.51	9.09	8.20	4.50
	24	1.02	0.99	1.52	3.16	10.37	3.90	2.61	8.65	0.99	3.34	9.18	8.22	4.50
	48	1.02	0.99	1.53	3.29	10.41	4.06	2.62	8.64	0.98	3.35	9.17	8.25	4.53
	96	1.02	0.99	1.59	3.60	10.40	4.84	2.63	9.10	1.01	3.36	9.29	8.30	4.68
CS	12	24.50	24.47	28.70	29.33	35.20	29.83	26.15	32.18	24.44	26.89	33.98	34.71	29.20
	24	24.37	24.45	28.62	29.41	34.92	29.88	25.98	32.20	24.34	26.83	33.56	33.68	29.02
	48	24.30	24.37	28.72	29.42	35.16	29.78	25.83	32.19	24.26	26.94	33.44	33.34	28.98
	96	24.36	24.37	28.56	29.57	35.63	29.97	25.79	32.20	24.29	26.77	33.44	33.10	29.00
DE	12	2.00	2.03	2.16	3.68	10.98	4.70	3.53	10.34	1.74	4.14	35.50	35.32	9.68
	24	1.95	1.99	2.11	3.65	11.20	4.12	3.54	9.36	1.70	4.28	32.82	31.37	9.01
	48	1.90	1.98	2.11	3.75	11.19	4.52	3.52	9.36	1.68	4.02	21.71	19.56	7.11
	96	1.83	2.00	2.13	3.97	11.25	5.44	3.49	9.90	1.68	4.10	9.97	9.08	5.40
FF	12	15.77	15.76	19.81	20.41	26.91	19.82	10.01	23.54	15.67	18.03	24.23	24.00	19.50
	24	30.64	30.65	34.76	35.42	41.97	34.75	16.60	38.67	30.58	33.15	39.26	38.81	33.77
	48	62.84	60.68	66.35	65.39	71.82	64.74	29.71	68.74	60.72	63.48	69.08	68.69	62.69
	96	118.96	118.91	122.99	123.56	129.87	122.98	55.15	126.88	118.86	121.27	127.10	126.84	117.78
GS	12	51.53	51.83	56.35	57.18	63.36	56.82	53.47	59.68	50.24	53.20	59.41	59.25	56.03
	24	106.75	108.12	112.49	112.54	117.82	109.98	107.82	113.98	106.07	108.48	113.99	113.08	110.93
	48	190.92	191.36	195.50	195.08	203.65	194.40	192.15	198.14	190.26	193.25	198.60	197.42	195.06
	96	383.60	384.10	387.67	388.08	396.93	387.45	385.62	391.97	383.77	386.76	391.12	390.42	388.12
PSO	12	12.23	7.59	10.07	13.95	19.74	14.59	7.74	12.95	18.59	14.59	21.83	21.69	14.63
	24	8.32	6.88	6.15	8.88	18.34	10.32	7.06	12.87	5.17	8.05	17.15	13.61	10.23
	48	7.65	5.84	6.07	8.36	16.48	10.02	6.79	12.85	5.04	7.41	14.45	12.68	9.47
	96	7.01	5.38	5.83	8.25	16.16	9.92	6.56	12.82	4.95	7.31	13.07	12.04	9.11
TLBO	12	4.00	3.92	4.48	7.46	14.90	8.96	5.42	12.53	5.41	6.19	12.21	8.83	7.86
	24	4.11	3.97	4.53	7.59	14.61	9.13	5.48	12.62	4.55	6.31	12.17	9.52	7.88
	48	4.10	3.99	4.54	7.58	14.66	9.25	5.49	12.60	4.54	6.30	12.17	10.36	7.96
	96	4.06	3.99	4.48	7.51	14.61	9.09	5.49	12.38	5.26	6.23	11.99	10.76	7.99



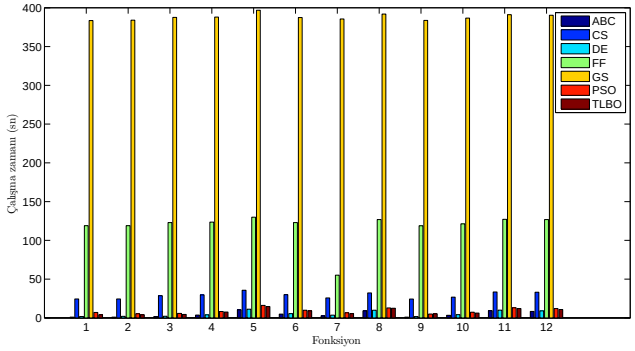
(a)



(b)

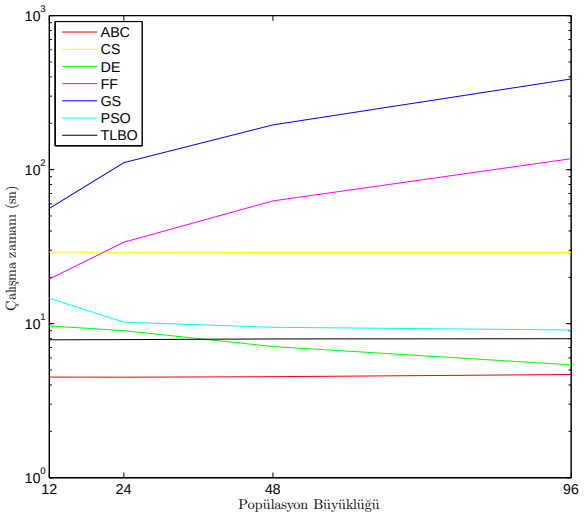


(c)



(d)

Şekil 5.12. Farklı popülasyon büyüklükleri ile koşulan ABC, CS, DE, FF, GS, PSO ve TLBO algoritmalarının test fonksiyonları için çalışma süreleri a)NP=12, b)NP=24, c)NP=48, d)NP=96



Şekil 5.13. Farklı popülasyon büyüklükleri ile koşulan ABC, CS, DE, FF, GS, PSO ve TLBO algoritmalarının ortalama çalışma süreleri

Tablo 5.31. ABC, CS, DE, FF, GS, PSO ve TLBO algoritmalarının hesaplama karmaşıklık değerleri

NP	Algoritma						
	ABC	CS	DE	FF	GS	PSO	TLBO
12	0.6371	7.1969	0.9550	4.4342	13.6544	1.5693	1.1941
24	0.6508	7.2083	0.9100	8.2883	26.4670	1.6149	1.1937
48	0.6599	7.2879	0.8186	16.1328	52.8427	1.6371	1.2056
96	0.7053	7.3670	0.7053	32.3220	105.0141	1.7057	1.2178

Algoritmaların hesaplama karmaşıklık değerlerinin verildiği Tablo 5.31 incelendiğinde, GS ve FF algoritmalarında popülasyon büyüklüğünün hesaplama karmaşıklık değerlerini oldukça etkilediği açıkca görülmektedir. GS ve FF algoritmalarındaki bu etki yeni çözüm üretme mekanizmalarından kaynaklanmaktadır. Yeni çözüm üretilirken GS algoritması popülasyondaki tüm bireylerden faydalanmaktadır, FF algoritması ise popülasyonda kendisinden daha iyi tüm bireylerden faydalanmaktadır. Dolayısı ile popülasyon boyutu bu algoritmaların çalışma zamanları için oldukça önem arz etmektedir. Sonuçlardan ayrıca FF ve GS algoritmaları ile orta ve büyük ölçekli problem çözümlerinde çok uzun süren çalışma zamanlarına ihtiyaç duyulacağı söylenebilir.

5.6.2. Alt Popülasyon Sayısının Çözüm Kalitesine ve Çalışma Zamanına Etkisi

PABC, PCS, PDE, PFF, PGS, PPSO, ve PTLBO algoritmalarının performans sonuçları sırası ile Tablo 5.32 - 5.38'de, paralel hızlanma grafikleri sırası ile Şekil 5.14 - 5.20 verilmiştir. Algoritmaların paralel modellerinin anlamlı bir karşılaştırma yapılabilmesi için de Tablo 5.39 ve Tablo 5.40 düzenlenmiştir. Tablo 5.39'da paralel algoritmalar her bir test fonksiyonu için göstermiş oldukları performansa göre sıralanmaktadır. Tablo 5.40'da ise Tablo 5.39'daki sıralama değerlerinin toplamı gösterilmekte ve paralel algoritmaların genel sıralaması verilmektedir. Ayrıca, algoritmaların detaylı çalışma zamanları Tablo 5.41'de verilmiştir.

Tablo 5.32 - 5.38'daki sonuçlar incelendiğinde alt popülasyon sayısının algoritmaların çözüm kalitesine etkisinin sınırlı olduğu görülmüştür. Diğer taraftan algoritmaların çözüm kalitelerine göre sıralandığı Tablo 5.39 ve Tablo 5.40'daki sonuçlardan seri algoritmalarda olduğu gibi PABC ve PDE algoritmalarının çözüm kalitelerinin daha üstün olduğu görülmektedir.

Tablo 5.41'deki çalışma zamanları ve Şekil 5.14 - 5.20'deki grafikler incelendiğinde paralel modelin tüm algoritmalar için zamansal kazanç sağladığı görülmektedir. Diğer taraftan PABC, PCS, PDE, PPSO ve PTLBO algoritmalarında hızlanma hemen hemen doğrusal iken, PFF ve PGS algoritmalarından süper doğrusal hızlanma elde edilmiştir. Paralel modeller birbirleri ile karşılaştırıldıklarında en hızlı algoritma PABC ve PDE algoritması olmuştur.

Tablo 5.32. Farklı CPU sayıları için PABC deneysel sonuçları

Fonksiyon	İşlemci Sayısı							
	2		4		8		16	
	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.
f_1	3.98E-13	1.76E-13	3.98E-13	1.86E-13	5.12E-13	1.37E-13	3.98E-13	1.95E-13
f_2	4.53E+01	1.25E+01	2.47E+01	1.21E+01	7.81E+00	8.90E+00	1.09E+00	3.42E+00
f_3	8.19E+00	1.06E+01	9.11E+00	1.19E+01	1.05E+01	1.06E+01	5.18E+01	5.79E+01
f_4	2.57E-02	1.38E-01	8.53E-13	1.29E-13	8.53E-13	1.49E-13	7.96E-13	3.02E-13
f_5	2.84E-13	3.67E-14	2.84E-13	4.61E-14	3.13E-13	5.85E-14	3.13E-13	4.76E-14
f_6	1.06E-09	3.92E-10	1.35E-09	4.65E-10	3.29E-09	1.65E-09	1.17E-09	1.07E-09
f_7	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
f_8	2.42E-01	3.24E-02	1.93E-01	4.51E-02	1.79E-01	4.26E-02	1.45E-01	5.60E-02
f_9	6.80E-12	1.76E-12	1.29E-11	3.48E-12	4.45E-11	1.97E-11	9.58E-11	2.85E-11
f_{10}	3.25E-04	1.96E-04	4.15E-04	2.60E-04	2.01E-03	2.55E-03	9.46E-01	2.35E+00
f_{11}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
f_{12}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00

Tablo 5.33. Farklı CPU sayıları için PCS deneysel sonuçları

Fonksiyon	İşlemci Sayısı							
	2		4		8		16	
	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.
f_1	1.76E-12	3.61E-12	1.14E-13	2.85E-13	6.25E-13	2.30E-13	6.50E-07	3.50E-06
f_2	9.72E+00	1.34E+01	1.33E+01	1.81E+01	1.71E+01	1.77E+01	5.75E+01	1.69E+01
f_3	1.26E+02	4.26E+01	1.11E+02	3.63E+01	2.11E+02	7.63E+01	1.63E+02	1.54E+02
f_4	3.62E+02	9.35E+01	2.38E+02	1.50E+02	9.42E+01	9.34E+01	2.38E+02	1.36E+02
f_5	1.86E-08	8.41E-08	4.93E-04	1.84E-03	2.05E-03	4.93E-03	1.47E-06	7.87E-06
f_6	3.41E+00	7.60E+00	7.87E-01	3.46E+00	4.45E-01	8.09E-01	3.01E+00	4.98E+00
f_7	1.00E-01	3.00E-01	3.00E-01	5.86E-01	9.33E-01	3.21E+00	3.57E+00	1.11E+01
f_8	1.28E-01	4.36E-02	1.29E-01	8.01E-02	1.57E-01	1.30E-01	1.02E-01	1.48E-01
f_9	8.08E-08	6.86E-08	7.44E-12	9.44E-12	1.39E-12	3.32E-12	1.87E-05	9.69E-05
f_{10}	2.15E+00	3.77E+00	1.09E+00	2.27E+00	2.28E+00	5.80E+00	5.00E-01	1.72E-09
f_{11}	9.01E-01	1.04E+00	3.50E-01	8.01E-01	3.67E-02	1.98E-01	1.76E-12	7.63E-12
f_{12}	8.86E+00	1.72E+01	9.38E+00	2.41E+01	7.32E-04	2.74E-03	1.52E-02	4.39E-02

Tablo 5.34. Farklı CPU sayıları için PDE deneysel sonuçları

Fonksiyon	İşlemci Sayısı							
	2		4		8		16	
	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.
f_1	5.68E-14	1.14E-13	5.68E-14	1.14E-13	5.68E-14	8.99E-14	5.68E-14	6.23E-14
f_2	2.21E+01	1.92E+00	1.64E+01	2.05E+00	1.12E+01	3.11E+00	2.77E+00	2.49E+00
f_3	1.82E+02	2.89E+01	1.61E+02	3.16E+01	1.43E+02	3.08E+01	6.05E+01	6.52E+01
f_4	1.14E-13	1.56E-13	6.73E-03	3.63E-02	5.68E-14	1.46E-13	5.68E-14	9.34E-14
f_5	1.14E-13	3.88E-13	2.84E-14	6.67E-14	2.84E-14	5.52E-14	2.84E-14	4.31E-14
f_6	2.99E-09	4.02E-10	2.40E-09	5.77E-10	1.40E-09	6.72E-10	2.93E-10	4.69E-10
f_7	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
f_8	1.64E-01	1.96E-02	1.52E-01	2.69E-02	1.58E-01	2.55E-02	1.05E-01	4.41E-02
f_9	1.66E-09	1.48E-10	1.22E-09	2.75E-10	5.82E-10	2.56E-10	8.60E-11	1.56E-10
f_{10}	6.90E-01	5.90E-01	6.68E-01	4.72E-01	6.52E-01	7.33E-01	8.68E-01	1.35E+00
f_{11}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
f_{12}	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00

Tablo 5.35. Farklı CPU sayıları için PFF deneysel sonuçları

Fonksiyon	İşlemci Sayısı							
	2		4		8		16	
	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.
f_1	3.76E-03	4.44E-04	3.53E-03	4.11E-04	3.45E-03	4.51E-04	3.23E-03	4.28E-04
f_2	9.92E-02	2.57E-02	1.26E-01	4.78E-02	1.39E+00	5.84E-01	2.16E+01	6.48E+00
f_3	5.06E+02	1.43E+03	9.97E+01	1.91E+02	9.76E+01	1.92E+02	6.94E+01	1.67E+02
f_4	1.43E+02	1.33E+01	1.70E+02	1.93E+01	2.31E+02	1.97E+01	3.63E+02	2.86E+01
f_5	2.50E-03	6.81E-04	2.03E-03	6.24E-04	1.71E-03	5.40E-04	1.54E-03	4.09E-04
f_6	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
f_7	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
f_8	4.41E-03	1.76E-03	4.93E-03	1.57E-03	8.99E-03	6.86E-03	1.25E-01	1.77E-01
f_9	1.84E-01	6.44E-02	1.98E-01	7.27E-02	2.05E-01	7.27E-02	2.71E-01	2.15E-01
f_{10}	1.16E+00	1.56E+00	7.07E-01	3.45E-01	7.01E-01	2.33E-01	7.21E-01	4.30E-01
f_{11}	3.54E-06	4.65E-07	3.35E-06	5.10E-07	3.16E-06	6.35E-07	3.24E+00	7.22E-01
f_{12}	1.90E-04	4.56E-05	1.69E-04	4.13E-05	1.58E-04	3.39E-05	4.81E-04	1.99E-03

Tablo 5.36. Farklı CPU sayıları için PGS deneysel sonuçları

Fonksiyon	İşlemci Sayısı							
	2		4		8		16	
	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.
f_1	3.47E+00	1.70E+01	1.11E+02	2.58E+02	1.72E+01	8.08E+01	9.00E+02	1.82E+03
f_2	6.93E+01	2.53E+00	7.14E+01	3.89E+00	7.19E+01	1.41E+01	7.56E+01	2.94E+00
f_3	9.61E+04	8.36E+04	6.18E+05	1.93E+06	6.74E+05	2.26E+06	1.55E+05	5.50E+05
f_4	6.50E+01	1.02E+01	2.45E+02	7.84E+01	1.15E+02	3.75E+01	1.78E+02	3.58E+01
f_5	1.53E+03	3.06E+02	1.41E+03	2.39E+02	1.31E+03	8.10E+01	1.33E+03	1.43E+02
f_6	1.18E-08	1.89E-09	1.43E+00	2.99E+00	7.43E-01	2.15E+00	4.10E+00	7.18E+00
f_7	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
f_8	5.57E-02	3.11E-02	3.35E+00	1.34E+01	6.13E-02	5.28E-02	1.42E-01	1.63E-01
f_9	1.86E-07	4.78E-08	1.08E-03	3.92E-03	3.74E-03	2.01E-02	1.00E-02	5.39E-02
f_{10}	4.91E-01	9.42E-02	1.19E+01	8.48E+00	8.95E+00	2.07E+01	4.83E-01	8.98E-02
f_{11}	0.00E+00	0.00E+00	1.71E-01	1.74E-01	1.59E+00	1.18E+00	3.01E-01	7.66E-01
f_{12}	7.32E-04	2.74E-03	1.10E+00	1.94E+00	3.83E+00	5.86E+00	1.40E+00	3.56E+00

Tablo 5.37. Farklı CPU sayıları için PPSO deneysel sonuçları

Fonksiyon	İşlemci Sayısı							
	2		4		8		16	
	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.
f_1	6.31E+02	6.27E+02	4.13E+01	4.82E+01	6.75E+00	1.38E+01	6.69E+00	7.09E+00
f_2	1.72E+01	1.07E+01	1.72E+01	4.70E+00	2.20E+01	4.11E+00	3.52E+01	4.59E+00
f_3	1.35E+07	5.23E+07	1.91E+02	5.51E+01	3.71E+02	1.16E+02	1.36E+04	8.20E+03
f_4	7.22E+02	5.42E+01	6.94E+02	4.58E+01	7.40E+02	3.63E+01	9.02E+02	5.82E+01
f_5	7.99E+00	1.07E+01	1.28E+00	8.74E-01	3.36E-01	3.38E-01	7.46E-01	2.57E-01
f_6	1.26E+01	7.32E+00	1.28E+01	7.52E+00	1.64E+01	5.59E+00	1.98E+01	1.41E-01
f_7	2.90E+00	6.02E+00	6.67E-01	1.11E+00	1.63E+00	1.38E+00	4.90E+00	2.61E+00
f_8	4.28E-02	9.24E-03	7.10E-02	2.11E-02	9.35E-02	2.42E-02	1.37E-01	3.17E-02
f_9	8.58E-13	5.44E-13	1.99E-10	1.26E-10	8.97E-08	4.18E-08	1.99E-04	1.35E-04
f_{10}	3.10E+00	5.31E+00	5.32E+00	5.41E+00	1.93E+01	7.84E+00	9.12E+01	2.24E+01
f_{11}	6.22E-03	2.33E-02	1.64E-08	5.36E-08	3.10E-01	3.75E-01	3.30E+00	1.49E+00
f_{12}	1.83E-03	4.09E-03	1.07E-10	2.33E-10	1.78E-04	4.26E-04	2.66E+01	1.63E+01

Tablo 5.38. Farklı CPU sayıları için PTLBO deneysel sonuçları

Fonksiyon	İşlemci Sayısı							
	2		4		8		16	
	Ort.	Std.	Ort.	Std.	Ort.	Std.	Ort.	Std.
f_1	7.07E+02	1.72E+03	6.84E+00	2.42E+01	1.34E+00	5.34E+00	6.92E+01	1.05E+02
f_2	8.97E+01	2.13E+00	9.02E+01	2.29E+00	9.23E+01	1.66E+00	9.43E+01	1.52E+00
f_3	2.72E+07	5.08E+07	2.44E+05	7.74E+05	2.28E+04	4.62E+04	1.19E+06	1.58E+06
f_4	7.33E+02	4.16E+01	8.30E+02	4.57E+01	1.03E+03	5.87E+01	1.18E+03	7.89E+01
f_5	5.82E+00	8.08E+00	9.72E-01	1.29E+00	5.02E-01	6.01E-01	2.06E+00	2.36E+00
f_6	1.80E+01	5.37E-01	1.86E+01	4.82E-01	1.94E+01	2.14E-01	1.96E+01	1.71E+00
f_7	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
f_8	1.87E-04	5.94E-05	1.98E-04	7.45E-05	2.05E-04	1.03E-04	1.51E-04	7.32E-05
f_9	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
f_{10}	5.00E-01	1.03E-16	5.00E-01	4.05E-17	5.00E-01	4.30E-17	5.00E-01	3.51E-17
f_{11}	1.47E-03	5.56E-03	1.31E-04	4.31E-04	5.41E-06	2.71E-05	5.38E-04	7.21E-04
f_{12}	6.04E-01	4.77E-01	1.41E+00	1.65E+00	3.70E+00	2.60E+00	7.67E+00	1.12E+00

Tablo 5.39. PABC, PCS, PDE, PFF, PGS, PPSO ve PTLBO algoritmalarının performans sıralaması

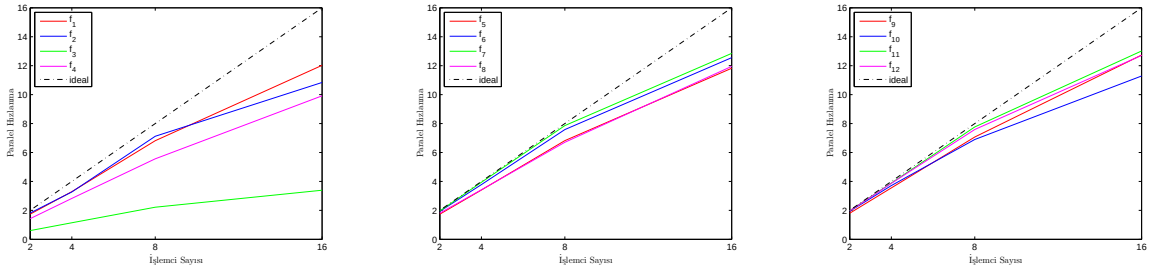
CPU Sayısı	Algoritma	Fonksiyon											
		f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}
2	PABC	2	5	1	2	2	2	1	7	3	1	1	1
	PCS	3	2	2	5	3	5	2	5	5	6	5	6
	PDE	1	4	3	1	1	3	1	6	4	4	1	1
	PFF	4	1	4	4	4	1	1	2	7	5	2	2
	PGS	5	6	5	3	7	4	1	4	6	2	1	3
	PPSO	6	3	6	6	6	6	3	3	2	7	4	4
	PTLBO	7	7	7	7	5	7	1	1	1	3	3	5
4	PABC	3	5	1	2	2	2	1	6	3	1	1	1
	PCS	2	2	3	4	3	4	2	4	2	5	6	6
	PDE	1	3	4	1	1	3	1	5	5	3	1	1
	PFF	4	1	2	3	4	1	1	2	7	4	3	3
	PGS	7	6	7	5	7	5	1	7	6	7	5	4
	PPSO	6	4	5	6	6	6	3	3	4	6	2	2
	PTLBO	5	7	6	7	5	7	1	1	1	2	4	5
8	PABC	2	2	1	2	2	3	1	7	3	1	1	1
	PCS	3	4	4	3	4	4	2	5	2	5	4	4
	PDE	1	3	3	1	1	2	1	6	4	3	1	1
	PFF	4	1	2	5	3	1	1	2	7	4	2	2
	PGS	7	6	7	4	7	5	1	3	6	6	6	6
	PPSO	6	5	5	6	5	6	3	4	5	7	5	3
	PTLBO	5	7	6	7	6	7	1	1	1	2	3	5
16	PABC	2	1	1	2	2	3	1	7	3	5	1	1
	PCS	3	5	4	4	3	4	2	2	4	2	2	3
	PDE	1	2	2	1	1	2	1	3	2	4	1	1
	PFF	4	3	3	5	4	1	1	4	7	3	5	2
	PGS	7	6	6	3	7	5	1	6	6	1	4	4
	PPSO	5	4	5	6	5	7	3	5	5	6	6	6
	PTLBO	6	7	7	7	6	6	1	1	1	2	3	5

Tablo 5.40. PABC, PCS, PDE, PFF, PGS, PPSO ve PTLBO algoritmalarının performans sıralama değerleri toplamı ve genel performans sıralamaları

CPU Sayısı		Algoritma						
		PABC	PCS	PDE	PFF	PGS	PPSO	PTLBO
2	Toplam	28	49	30	37	47	56	54
	Genel Performans Sırası	1	5	2	3	4	7	6
4	Toplam	28	43	29	35	67	53	51
	Genel Performans Sırası	1	4	2	3	7	6	5
8	Toplam	26	44	27	34	64	60	51
	Genel Performans Sırası	1	4	2	3	7	6	5
16	Toplam	29	38	21	42	56	63	52
	Genel Performans Sırası	2	3	1	4	6	7	5

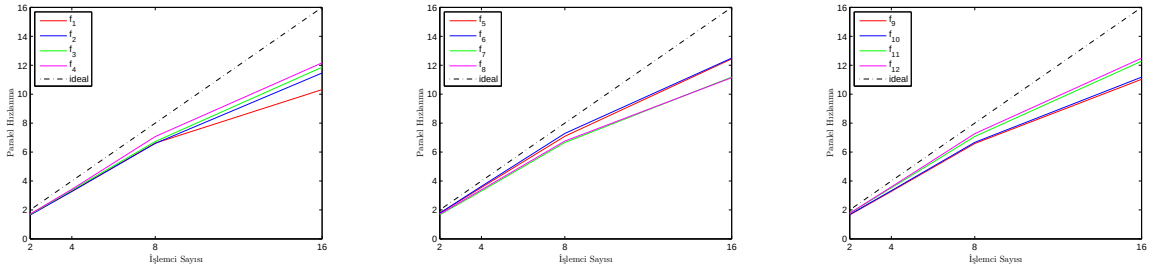
Tablo 5.41. Farklı CPU sayıları ile koşulan PABC, PCS, PDE, PFF, PGS, PPSO ve PTLBO algoritmalarının çalışma süreleri

Algoritma	CPU Sayısı	Fonksiyon												Avg.
		f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	
ABC	1	1.02	0.99	1.59	3.60	10.40	4.84	2.63	9.10	1.01	3.36	9.29	8.30	4.68
PABC	2	0.59	0.54	2.64	2.52	5.99	2.50	1.34	5.02	0.56	1.73	4.70	4.23	2.70
	4	0.31	0.30	1.39	1.27	3.05	1.28	0.67	2.66	0.28	0.90	2.39	2.15	1.39
	8	0.15	0.14	0.72	0.65	1.53	0.64	0.33	1.36	0.14	0.49	1.20	1.09	0.70
	16	0.09	0.09	0.47	0.36	0.88	0.39	0.20	0.76	0.08	0.30	0.71	0.65	0.42
CS	1	24.36	24.37	28.56	29.57	35.63	29.97	25.79	32.20	24.29	26.77	33.44	33.10	29.00
PCS	2	14.65	14.71	16.83	17.19	20.06	16.79	15.42	18.73	14.65	15.89	18.94	18.65	16.88
	4	7.42	7.43	8.49	8.63	10.03	8.25	7.78	9.49	7.39	8.03	9.46	9.22	8.47
	8	3.68	3.69	4.24	4.18	5.02	4.11	3.87	4.76	3.68	4.01	4.72	4.55	4.21
	16	2.36	2.13	2.41	2.43	2.86	2.40	2.31	2.89	2.20	2.39	2.72	2.65	2.48
DE	1	1.83	2.00	2.13	3.97	11.25	5.44	3.49	9.90	1.68	4.10	9.97	9.08	5.40
PDE	2	0.94	1.04	3.06	2.70	6.42	2.79	1.77	5.22	0.87	2.18	5.07	4.63	3.06
	4	0.47	0.51	1.47	1.36	3.19	1.37	0.89	2.62	0.43	1.03	2.52	2.28	1.51
	8	0.24	0.26	0.76	0.68	1.60	0.69	0.45	1.26	0.22	0.55	1.28	1.16	0.76
	16	0.13	0.14	0.40	0.35	0.82	0.33	0.23	0.90	0.12	0.29	0.73	0.61	0.42
FA	1	118.96	118.91	122.99	123.56	129.87	122.98	55.15	126.88	118.86	121.27	127.10	126.84	117.78
PFF	2	55.76	55.84	57.96	58.45	61.60	58.13	26.46	60.10	55.90	57.09	60.27	60.12	55.64
	4	14.16	14.24	15.16	15.29	16.92	15.17	7.34	16.48	14.17	14.79	16.28	16.18	14.68
	8	3.64	3.64	4.15	4.21	5.04	4.14	2.14	4.74	3.60	3.92	4.74	4.71	4.06
	16	0.91	0.90	1.17	1.21	1.62	1.20	0.64	1.50	0.90	1.05	1.53	1.50	1.18
GSA	1	383.60	384.10	387.67	388.08	396.93	387.45	385.62	391.97	383.77	386.76	391.12	390.42	388.12
PGS	2	150.71	150.53	153.26	152.97	157.39	152.32	151.98	154.35	151.04	151.91	155.04	154.59	153.01
	4	37.83	37.84	38.90	39.07	41.09	38.88	38.25	39.94	37.77	38.42	39.85	39.70	38.96
	8	9.40	9.37	9.89	10.05	11.00	9.92	9.55	10.45	9.37	9.68	10.42	10.35	9.95
	16	4.49	4.47	5.01	5.08	6.10	5.10	4.65	5.49	4.48	4.80	5.63	5.63	5.08
PSO	1	7.01	5.38	5.83	8.25	16.16	9.92	6.56	12.82	4.95	7.31	13.07	12.04	9.11
PPSO	2	3.96	3.78	5.68	5.89	9.21	5.97	4.35	7.61	3.51	4.73	7.66	7.18	5.79
	4	1.90	1.96	2.83	3.03	4.57	3.02	2.23	3.86	1.77	2.39	3.86	3.66	2.92
	8	0.97	1.06	1.45	1.63	2.32	1.61	1.16	1.98	0.90	1.25	2.05	1.94	1.53
	16	0.57	0.73	1.23	1.35	1.30	1.36	0.61	1.05	0.46	1.16	1.59	1.56	1.08
TLBO	1	4.06	3.99	4.48	7.51	14.61	9.09	5.49	12.38	5.26	6.23	11.99	10.76	7.99
PTLBO	2	2.79	2.77	4.91	5.22	8.65	5.40	3.44	7.35	3.36	4.01	6.85	6.23	5.08
	4	1.41	1.39	2.47	2.67	4.42	2.77	1.73	3.87	1.66	2.00	3.52	3.07	2.58
	8	0.71	0.70	1.26	1.36	2.19	1.39	0.87	2.09	0.82	1.02	1.74	1.46	1.30
	16	0.42	0.44	0.74	0.78	1.34	0.78	0.56	1.28	0.48	0.57	0.98	0.76	0.76



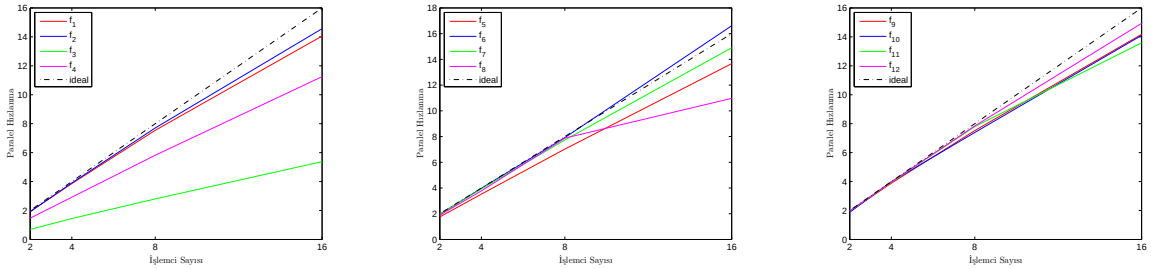
(a) (b) (c)

Şekil 5.14. PABC algoritması için paralel hızlanma grafiği



(a) (b) (c)

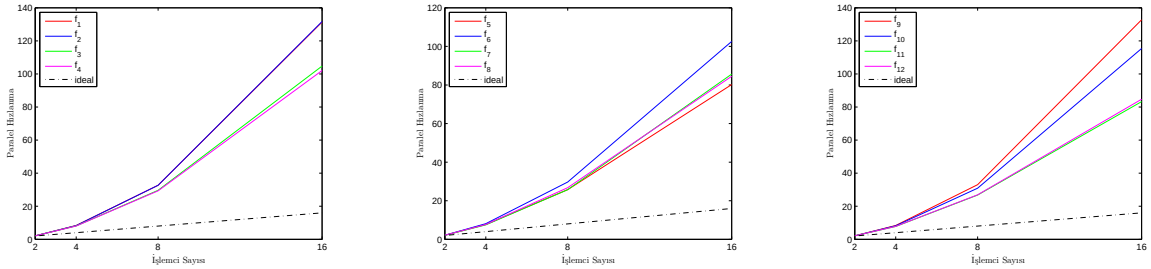
Şekil 5.15. PCS algoritması için paralel hızlanma grafiği



(a) (b) (c)

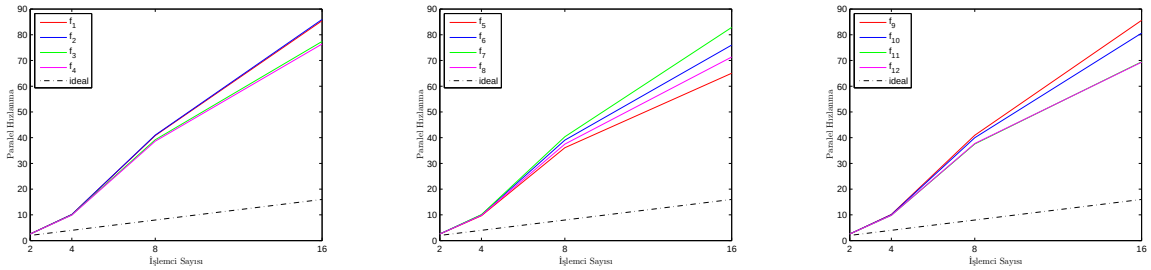
Şekil 5.16. PDE algoritması için paralel hızlanma grafiği

Bu uygulamada yeni ve popüler bazı popülasyon tabanlı sezgisel algoritmaların seri ve paralel modelleri kıyaslanmıştır. ABC, CS, DE, FF, GS, PSO ve TLBO olmak üzere yedi farklı sezgisel algoritma dikkate alınmıştır. Çalışma ile algoritmaların güçlü ve zayıf yönlerinin tespit edilmesi amaçlanmıştır. Çalışmada algoritmalarda popülasyon büyüklüğünün sadece performanslarını değil aynı zamanda çalışma zamanlarını da etkilediği gösterilmiştir. Paralel modellerle zamansal kazançlar sağlanabildiği, fakat FF ve GS gibi algoritmalarla büyük boyutlu problem çözümlerinin çok uzun süreler aldığı gözlemlenmiştir.



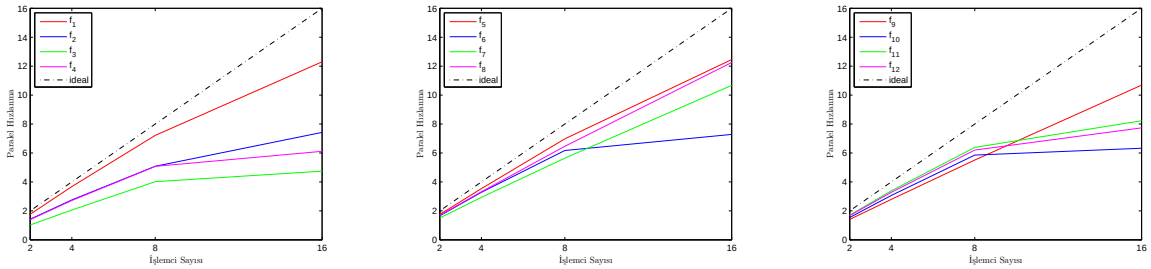
(a) (b) (c)

Şekil 5.17. PFF algoritması için paralel hızlanma grafiği



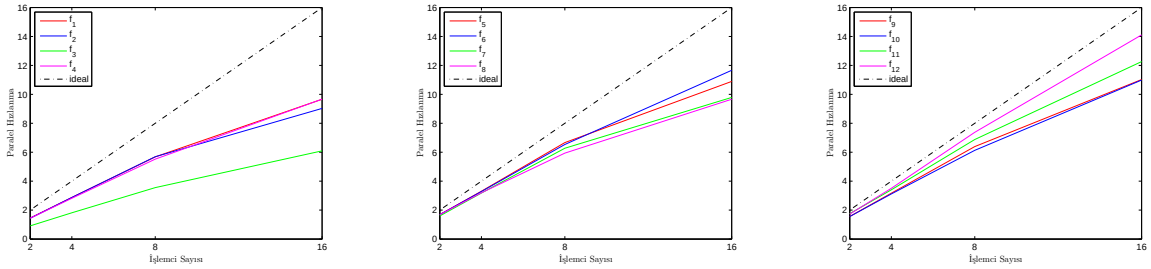
(a) (b) (c)

Şekil 5.18. PGS algoritması için paralel hızlanma grafiği



(a) (b) (c)

Şekil 5.19. PPSO algoritması için paralel hızlanma grafiği



(a) (b) (c)

Şekil 5.20. PTLBO algoritması için paralel hızlanma grafiği

5.7. DE Algoritmasının Paralel Strateji Portföy Yapısı

DE algoritması, sayısal problem çözümleri için geliştirilmiş, etkin, basit ve güçlü bir sezgisel algoritmadır. Birçok araştırma alanında karşılaşılan çeşitli problem türlerinde kullanılmaktadır. Algoritma yeni çözüm üretmek için farklı stratejiler kullanabilmekte ve algoritmaya özel kontrol parametreleri içermektedir. Dolayısı ile algoritmanın performansı seçilen stratejiye ve kontrol parametre değerlerine göre değişmektedir. Bu aşamada, araştırmacılar problem setlerine uygun strateji ve kontrol parametre değerlerini belirlemek için çok sayıda deneme yanılma yapmak zorunda kalmaktadırlar. Bu işlemler yüksek hesaplama maliyetlerini ve uzun çalışma sürelerini beraberinde getirmektedir. Bu dezavantajları gidermek ve problemlerin çözüm kalitelerini üst düzeye çıkarmak için strateji ve kontrol parametrelerinin uyarlanabilir olduğu bazı çalışmalar yapılmaktadır.

Bu çalışmanın amaçları, DE algoritmasının stratejiye olan performans bağımlılığını önlemek için yeni bir model önermek ve önerilen modelin paralel gerçekleştirimi ile paralel hesaplama sistemlerinin avantajlarından faydalanmaktır. Ayrıca birden fazla stratejinin aynı anda kullanılmasıyla arama sürecinde farklı komşulukların aranması ve daha kaliteli çözümler üretilmesi beklenmektedir. Bu amaçla popülasyonun tamamı küçük boyutlu alt popülasyonlara bölünmüş, her bir alt popülasyonun farklı bir mutasyon stratejisi ile çalışması sağlanmıştır. Bu alt popülasyonlar belirlenen aralıklarla birbirleri ile haberleşmekte, kendi strateji bilgilerini ve elde ettikleri en iyi çözüm vektörünü paylaşmaktadırlar. Dolayısı ile farklı stratejilerin avantajları tüm alt popülasyonlara yayılmaktadır. Alt popülasyon yaklaşımı popülasyon tabanlı sezgisel algoritmaların paralelleştirilme mimarilerinden coarse-grained modeli için oldukça uygundur. Bu model ile her bir alt popülasyonun farklı bir işlemcide çalışması sağlanarak, belirli kriterlere göre haberleştirilmektedir. Coarse-grained modeli ile yapılan çalışmalarda genel olarak alt popülasyonların tamamında aynı algoritmik kod çalışmakta ve haberleşme esnasında kaliteli çözüm vektörü ya da vektörleri paylaşılmaktadır. Bu model birçok sezgisel algoritmanın performansını önemli ölçüde iyileştirebilmektedir.

Önerilen modelin temelinde ise farklı işlemcilerde farklı stratejileri içeren algoritmik

kodun çalışması vardır. Çünkü DE algoritmasının sahip olduğu mutasyon stratejileri, farklı problem türlerinde farklı performans sergileyebilmektedirler. Farklı stratejiler ile çalışan işlemciler yine belirli aralıklarla bilgi paylaşımı yapmaktadırlar. Önerilen modelde bilgi paylaşımı esnasında işlemciler kaliteli çözüm vektörü ile beraber kendi mutasyon strateji bilgisini de bildirmektedir. İşlemciler kendilerindeki en iyi değerleri ve transfer olan en iyi değerleri karşılaştırarak en başarılı stratejiyi seçebilmektedir. Dolayısı ile paylaşım neticesinde başlangıçta farklı işlemcilerde çalışan farklı stratejiler, ilerleyen iterasyonlarda adaptif olarak başarılı stratejiye doğru yönelmektedir. Böylelikle DE algoritmasının seçilen mutasyon stratejine olan bağımlılığı ortadan kaldırılmakta ve daha geniş problem setinde başarılı çözümler üreten kararlı bir algoritma modeli ortaya konulmaktadır. Önerilen paralel algoritma modeli PDESP olarak adlandırılmıştır (Parallel Diferential Evolution with Strategy Portfolio, PDESP). Önerilen modelde DE/rand/1, DE/best/1, DE/rand-to-best/1, DE/rand/2 ve DE/best/2 olmak üzere beş farklı mutasyon strateji kullanılmıştır. Kullanılan bu stratejiler sırası ile DE_{S1}, DE_{S2}, DE_{S3}, DE_{S4} ve DE_{S5} olarak adlandırılmış ve fark vektörü formülleri sırası ile Eşitlik (5.2) - (5.6)'de verilmiştir.

- DE/rand/1 (DE_{S1})

$$v_i(t) = x_{r_1}(t) + F(x_{r_2}(t) - x_{r_3}(t)) \tag{5.2}$$

- DE/best/1 (DE_{S2})

$$v_i(t) = x_{best}(t) + F(x_{r_1}(t) - x_{r_2}(t)) \tag{5.3}$$

- DE/rand-to-best/1 (DE_{S3})

$$v_i(t) = x_i(t) + F(x_{best}(t) - x_i(t)) + F(x_{r_1}(t) - x_{r_2}(t)) \tag{5.4}$$

- DE/rand/2 (DE_{S4})

$$v_i(t) = x_{best}(t) + F(x_{r_1}(t) - x_{r_2}(t)) + F(x_{r_3}(t) - x_{r_4}(t)) \tag{5.5}$$

- DE/best/2 (DE_{S5})

$$v_i(t) = x_{r_1}(t) + F(x_{r_2}(t) - x_{r_3}(t)) + F(x_{r_4}(t) - x_{r_5}(t)) \tag{5.6}$$

Burada r_1, r_2, r_3, r_4, r_5, $[1, NP]$ aralığında her bir iterasyonda değişen birbirinden farklı tamsayılardır. F ölçekleme faktörü, $v_i(t)$ ise fark vektörüdür.

Yapılan bu çalışmada da, CEC2008 için özel olarak hazırlanmış zorlaştırılmış altı adet test fonksiyonu ile yaygın olarak kullanılan altı adet test fonksiyonu kullanılmıştır. Test fonksiyonları için problem boyutları 100, iterasyon sayısı 500000 alınmıştır. Bu değerler CEC2008 yarışması için tanımlanan özel değerlerdir. Çalışmalarda popülasyon boyutu 50 ve 100 olmak üzere iki farklı durum için incelenmiştir. Ayrıca algoritmaya özgü parametre olan ölçekleme faktörü ve çaprazlama oranı için sırası ile 0.5 ve 0.3 değerleri alınmıştır. NP=50 için Tablo 5.42 ve NP=100 için Tablo 5.43'de 30 koşma sonucu oluşan çözümlerin ortalama uygunluk ve standart sapma değerleri verilmiştir. Ayrıca seri ve paralel modellerin ortalama çalışma süreleri ve elde edilen hızlanma ve verimlilik değerleri Tablo 5.44'de verilmiştir.

Gerçekleştirilen bu uygulamadan iki farklı deneyim çıkarılmıştır. Bunlardan birincisi farklı mutasyon stratejileri ile çalışan DE ve önerilen PDESP algoritmalarının çözüm kaliteleri açısından karşılaştırılması, bir diğeri ise coarse-grained paralel modelinin sağladığı zamansal kazancın incelenmesi.

Tablo 5.42 ve Tablo 5.43'deki sonuçlardan popülasyon büyüklüğü 100 için farklı mutasyon stratejileri birbirleri ile karşılaştırıldığında f_1, f_7, f_{11} ve f_{12} fonksiyonlarında tüm stratejiler eşit, f_5, f_6, f_8 ve f_9 fonksiyonlarında strateji DE_{S2}, f_{10} fonksiyonunda strateji DE_{S4}, f_2 ve f_4 fonksiyonunda strateji DE_{S1} ve f_3 fonksiyonunda strateji DE_{S3} en başarılı çözümler üretmektedir. Benzer bir tablo popülasyon büyüklüğü 50 içinde geçerlidir. Görüldüğü üzere farklı stratejiler farklı problemler türlerinde daha başarılı olmaktadırlar. Önerilen PDESP algoritması dikkate alındığında ise, PDESP algoritmasının tüm stratejilere oranla daha kararlı bir algoritma olduğu görülmektedir. Sonuçlar tüm test fonksiyonlarında PDESP algoritmasının en iyi ya da en iyi çözüm üreten DE stratejisine çok yakın değerler ürettiğini göstermektedir.

Tablo 5.42. PDESP ve farklı mutasyon stratejileri ile koşulan DE algoritmalarının ortalama ve standart sapma değerleri, Ort.: En iyi değerlerin ortalamaları, Std.: En iyi değerlerin standart sapmaları ($NP = 50$)

Fonksiyon	f_1			f_2			f_3			f_4		
CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
DE_{S1}	4.73E-02	2.01E-01	1.9207	7.44E+00	4.87E-01	2.0083	1.56E+02	2.14E+01	2.1260	1.35E-02	5.04E-02	3.7987
DE_{S2}	3.23E-02	1.21E-01	1.9110	8.90E+01	9.93E+00	2.0353	1.51E+02	5.29E+01	2.2303	2.72E-01	5.20E-01	3.7607
DE_{S3}	0	0	1.9607	8.40E+01	6.22E+00	2.0373	7.12E+01	3.34E+01	2.2010	1.00E-01	2.98E-01	3.9447
DE_{S4}	2.74E-02	1.04E-01	1.9203	1.74E+01	2.33E+00	2.0073	1.33E+02	2.88E+01	2.1270	2.77E-02	8.95E-02	3.8593
DE_{S5}	0	0	1.8030	1.44E+01	8.05E-01	1.9740	1.65E+02	1.88E+01	2.1623	2.02E-02	6.06E-02	3.8700
PDESP	0	0	0.4093	7.45E+00	1.93E+00	0.4163	1.39E+02	3.64E+01	1.1510	6.73E-03	3.63E-02	1.0507
Fonksiyon	f_5			f_6			f_7			f_8		
CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
DE_{S1}	0	0	11.3637	0	0	4.6213	0	0	3.5203	9.41E-02	1.07E-02	9.3553
DE_{S2}	9.66E-03	5.20E-02	11.4040	0	0	4.6100	0	0	3.6267	8.70E-02	8.61E-03	9.5093
DE_{S3}	0	0	11.1790	0	0	4.6657	0	0	3.5137	9.06E-02	8.57E-03	9.3323
DE_{S4}	0	0	11.1130	0	0	4.6200	0	0	3.5003	1.09E-01	1.15E-02	9.7180
DE_{S5}	0	0	11.0567	3.98E-13	0	4.6970	0	0	3.3963	1.12E-01	1.10E-02	9.6057
PDESP	0	0	2.4483	0	0	0.9763	0	0	0.7300	2.78E-02	7.43E-03	2.4873
Fonksiyon	f_9			f_{10}			f_{11}			f_{12}		
CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
DE_{S1}	0	0	1.7033	5.18E-01	4.74E-02	4.1747	0	0	20.7630	0	0	18.6663
DE_{S2}	0	0	1.7267	1.28E+00	2.41E+00	4.1980	0	0	22.4257	0	0	20.1557
DE_{S3}	0	0	1.7250	1.91E+00	2.97E+00	4.1270	0	0	17.9563	0	0	15.8360
DE_{S4}	0	0	1.6873	5.02E-01	3.47E-03	3.9237	0	0	16.7840	0	0	14.4327
DE_{S5}	0	0	1.6677	5.20E-01	2.86E-02	4.0263	0	0	14.7993	0	0	12.4597
PDESP	0	0	0.3943	5.00E-01	9.54E-11	0.8920	0	0	4.1203	0	0	3.7660

Tablo 5.43. PDESP ve farklı mutasyon stratejileri ile koşulan DE algoritmalarının ortalama ve standart sapma değerleri, Ort.: En iyi değerlerin ortalamaları, Std.: En iyi değerlerin standart sapmaları ($NP = 100$)

Fonksiyon	f_1			f_2			f_3			f_4		
CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
DE_{S1}	0	0	1.8450	2.75E+01	1.03E+00	2.0237	2.05E+02	3.05E+01	2.1470	0	0	4.0027
DE_{S2}	0	0	1.8430	7.43E+01	9.35E+00	2.0373	1.72E+02	4.42E+01	2.2337	6.73E-03	3.63E-02	3.9817
DE_{S3}	0	0	1.8723	7.10E+01	6.14E+00	2.0583	1.33E+02	5.02E+01	2.1780	3.65E-05	1.31E-04	4.1947
DE_{S4}	0	0	1.8233	3.03E+01	3.07E+00	2.0253	1.83E+02	3.19E+01	2.1560	1.34E-10	3.36E-11	4.1090
DE_{S5}	5.97E-12	8.48E-13	1.7343	3.78E+01	1.64E+00	2.0000	2.21E+02	3.59E+01	2.2030	4.37E-09	1.10E-09	4.1520
PDESP	0	0	0.3773	2.71E+01	3.38E+00	0.4093	1.86E+02	4.51E+01	1.1267	0	0	1.1037
Fonksiyon	f_5			f_6			f_7			f_8		
CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
DE_{S1}	1.14E-13	1.89E-13	11.2850	1.02E-08	1.04E-09	5.4457	0	0	3.3937	1.75E-01	1.69E-02	9.3450
DE_{S2}	2.84E-14	6.81E-14	11.4453	9.32E-10	9.45E-11	5.3597	0	0	3.5950	1.62E-01	1.60E-02	10.1270
DE_{S3}	6.93E-12	1.95E-11	11.3843	1.57E-07	2.19E-08	5.6210	0	0	3.5480	1.69E-01	1.35E-02	9.3497
DE_{S4}	5.54E-11	2.88E-10	11.3203	1.91E-07	2.09E-08	5.5883	0	0	3.5297	2.04E-01	1.82E-02	9.3483
DE_{S5}	2.38E-08	9.81E-08	11.2937	1.11E-06	8.83E-08	5.6087	0	0	3.4207	2.21E-01	2.40E-02	9.3373
PDESP	2.42E-13	1.15E-12	2.5037	1.88E-08	7.87E-09	1.1313	0	0	0.7343	5.66E-02	1.54E-02	1.8147
Fonksiyon	f_9			f_{10}			f_{11}			f_{12}		
CPU Sayısı	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman	Ort.	Std.	Zaman
DE_{S1}	5.89E-09	5.22E-10	1.7023	7.10E-01	3.64E-01	4.1497	0	0	9.9773	0	0	9.0350
DE_{S2}	4.09E-10	4.66E-11	1.7240	1.22E+00	2.23E+00	4.1640	0	0	10.0180	0	0	8.9857
DE_{S3}	7.30E-08	4.92E-09	1.7370	1.18E+00	1.83E+00	3.9987	0	0	9.9270	0	0	9.1080
DE_{S4}	1.73E-07	1.63E-08	1.6943	5.18E-01	2.41E-02	3.9943	0	0	10.0583	0	0	9.1857
DE_{S5}	9.53E-07	5.48E-08	1.6737	8.61E-01	1.57E-01	4.0243	0	0	9.9910	0	0	9.2650
PDESP	0	0	0.3613	5.00E-01	3.45E-07	0.8390	0	0	1.9430	0	0	1.7703

Tablo 5.44. PDESP ve farklı mutasyon stratejileri ile koşulan DE algoritmalarının ortalama çalışma süreleri, hızlanma ve verimlilik değerleri

Fonksiyon	Popülasyon Büyüklüğü							
	50				100			
	Ort(DE)	PDESP	Hız.	Ver.	Ort(DE)	PDESP	Hız.	Ver.
f_1	1.9031	0.4093	4.65	0.93	1.8236	0.3773	4.83	0.97
f_2	2.0125	0.4163	4.83	0.97	2.0289	0.4093	4.96	0.99
f_3	2.1693	1.1510	1.88	0.38	2.1835	1.1267	1.94	0.39
f_4	3.8467	1.0507	3.66	0.73	4.0880	1.1037	3.70	0.74
f_5	11.2233	2.4483	4.58	0.92	11.3457	2.5037	4.53	0.91
f_6	4.6428	0.9763	4.76	0.95	5.5247	1.1313	4.88	0.98
f_7	3.5115	0.7300	4.81	0.96	3.4974	0.7343	4.76	0.95
f_8	9.5041	2.4873	3.82	0.76	9.5015	1.8147	5.24	1.05
f_9	1.7020	0.3943	4.32	0.86	1.7063	0.3613	4.72	0.94
f_{10}	4.0899	0.8920	4.59	0.92	4.0662	0.8390	4.85	0.97
f_{11}	18.5457	4.1203	4.50	0.90	9.9943	1.9430	5.14	1.03
f_{12}	16.3101	3.7660	4.33	0.87	9.1159	1.7703	5.15	1.03

Tablo 5.44'deki hızlanma ve verimlilik değerleri incelendiğinde, coarse-grained paralelleştirme modelinin seri modele oranla daha hızlı çözümler ürettiği de görülmektedir. Çalışmada paralel hesaplama sistemlerinin getirdiği avantajlardan faydalanarak doğrusala yakın bir hızlanma elde edilmiştir.

Bu uygulama ile literatürde oldukça yaygın olarak kullanılan DE algoritmasının mutasyon stratejisine olan performans bağımlılığının önlenebileceği vurgulanmaktadır. Önerilen modelde farklı stratejilerin bir arada kullanılması ile her birinin sahip olduğu avantajlardan faydalanılmaktadır. Bu çalışmada farklı mutasyon stratejileri detaylı analiz edilmemiş, yaygın kullanılan bazı stratejiler dikkate alınmıştır. Stratejilerin detaylı analiz edildiği ve özellikle birbirlerini tamamlayan stratejiler ile oluşturulan PDESP yapısı ile daha başarılı sonuçlar elde edilebileceği düşünülmektedir.

5.8. Sınıflandırma Problemi İçin Paralel ABC Algoritması

Sınıflandırma, veri madenciliği, istatistiksel veri analizi, veri sıkıştırma gibi çeşitli uygulamalar için önemli bir problemdir. Bu problemde verilerin sınıf içi benzerlik

maksimum, sınıflar arası benzerlik minimum olacak şekilde sınıflara ayrılması amaçlanır. Problemin çözümü için farklı bilimsel alanlarda geliştirilen çeşitli çalışmalar bulunmaktadır. Özellikle klasik ve geleneksel yöntemlerin yüksek hesaplama maliyeti, yerel en iyiye takılmaları gibi dezavantajları sebebi ile yapay sinir ağları [173–175], evrimsel hesaplama [176–183] gibi esnek hesaplama yöntemleri yaygın olarak tercih edilebilmektedir.

ABC algoritmasının sınıflandırma probleminde kullanılmasına yönelik çeşitli çalışmalar bulunmaktadır [184–187]. Bu çalışmaların tamamı algoritmanın temel yada geliştirilmiş seri modelleri ile gerçekleştirilmiştir. Önerilen yaklaşımda ise ABC algoritmasının paralel modeli ile yapay sinir ağlarının eğitilmesi ve bazı sınıflandırma problemlerinin çözülmesi amaçlanmıştır. Uygulamada ABC algoritmasının coarse-grained paralelleştirme modeli gerçekleştirilmiştir (Parallel Artificial Bee Colony, PABC).

PABC algoritmasın da yapay sinir ağlarına ait bağlantı ağırlık değerleri ve eşik değerleri problem parametreleri olarak düşünülür. Algoritmaya ait yeni çözüm üretme operatörleri ile bu parametreler değiştirilerek yapay sinir ağının ürettiği hata değeri düşürülmeye çalışılmaktadır. Yapılan çalışmada YSA modellerinden çok katmanlı ileri beslemeli yapay sinir ağı modeli, transfer fonksiyonlarından sigmoid transfer fonksiyonu, hata değeri hesaplanması için ağın çıkış değeri ve istenen çıkış değerinin farkının karesinin alındığı ortalama karesel hata (Mean Sequared Error, MSE) fonksiyonu kullanılmıştır. Eşitlik (5.7)'de kullanılan hata fonksiyonunun matematiksel formülü verilmiştir.

$$E(w(t)) = \frac{1}{n} \sum_{j=1}^{n} \sum_{k=1}^{K} (o_k - d_k)^2 \tag{5.7}$$

Burada, $E(w(t))$ $t$inci iterasyondaki hata değeri, $w(t)$ $t$inci iterasyondaki ağırlıklar ve eşik değerleri, d_k istenen çıkış, o_k hesaplanan network çıkışıdır. Burada en iyi ağırlık ve eşik değerlerinin bulunarak $E(w(t))$ hata değerinin minimize edilmesi de optimizasyon işlemidir.

Algoritmanın performansını değerlendirmek için ise Eşitlik (5.8) ve Eşitlik (5.9)'da

verilen sırası ile eğitim ve test sınıflandırma hata yüzdeleri (Classification Error Percentage, CEP) kullanılmıştır.

$$CEP_{Egitim} = 100x\frac{\text{hatalı sınıflandırılan eğitim örnek sayısı}}{\text{toplam eğitim veri sayısı}} \tag{5.8}$$

$$CEP_{Test} = 100x\frac{\text{hatalı sınıflandırılan test örnek sayısı}}{\text{toplam test veri sayısı}} \tag{5.9}$$

Gerçekleştirilen PABC algoritmasının performansı ve çalışma zamanının test edilmesi için literatürde yaygın olarak kullanılan 5 farklı veri kümesi kullanılmıştır. Çalışmada dikkate alınan veri kümeleri aşağıda kısaca tarif edilerek özellikleri Tablo 5.45'de verilmiştir.

Balance veri seti psikolojik deney sonuçlarını modellemek için oluşturulmuştur. Veri seti 469 eğitim ve 156 test olmak üzere 625 örnekten oluşur. Cancer-Int meme kanseri tanısı için oluşturulmuş bir veri kümesidir. Veri seti toplam 699 örnek için 9 giriş ve 2 sınıf bilgisi içerir. Glass veri seti cam türlerini sınıflandırmak için kullanılır. Iris veri seti Setosa, Versicolor ve Virginica olmak üzere Iris çiçek sınıflarını tanımlamak için 150 örnekten oluşur. Bu üç sınıfın her bir için 50 örnek bulunur. Verilerde sepal uzunluk, sepal genişlik, petal uzunluk ve petal genişlik olmak üzere çiçeklerin 4 özelliği verilmektedir. Wine veri seti, 3 farklı üreticiden alınan toplam 178 şarap örneğinin kimyasal analizinden oluşturulmuştur. Kimyasal analiz sonucu elde edilen 13 farklı değer bu 3 sınıfı belirlemek için kullanılır. Veri kümeleri hakkında detaylı bilgi [188]'de bulunabilir.

Tablo 5.45. Deneysel çalışmalarda kullanılan sınıflandırma problemlerinin özellikleri

Problem	Veri sayısı	Eğitim verisi	Test verisi	Giriş sayısı	Sınıf sayısı
Balance	625	469	156	4	3
Cancer-Int	699	524	175	9	2
Glass	214	161	53	9	2
Iris	150	112	38	4	3
Wine	178	133	45	13	3

Yapay sinir ağının başarısı ve çalışma zamanı büyük ölçüde yapısına bağlıdır. Gerçekleştirilen çok katmanlı ileri beslemeli yapay sinir ağı modelinde tek bir ara katman kullanılmıştır. Bu ara katmanda 2, 5 ve 8 nöron olmak üzere üç farklı durum analiz edilmiştir. ABC ve PABC algoritmaları için, popülasyon büyüklüğü, limit ve toplam değerlendirme sayısı sırasıyla 120, 1000 ve 20000 olarak seçilmiştir. Ayrıca istatistiksel bilgilerin oluşturulabilmesi için gerçekleştirilen 30 koşmanın her birinde, YSA eğitilmeden önce veriler rastgele olarak karıştırılmıştır. ABC ve PABC algoritmaları ile eğitilen farklı YSA yapılarında elde edilen hızlanma değerleri Tablo 5.46'de ve ortalama sınıflandırma hata değerleri Tablo 5.47'de verilmiştir.

Tablo 5.46. ABC ve PABC algoritmaları ile eğitilen farklı YSA yapılarının ortalama çalışma zamanları ve elde edilen hızlanma değerleri

		ABC	PABC			
		1 CPU	10 CPU		20 CPU	
Problem	YSA Yapısı	Zaman	Zaman	Hızlanma	Zaman	Hızlanma
Balance	4-2-3	41.0667	4.2137	9.75	2.1673	18.95
	4-5-3	86.6643	8.8953	9.74	4.5430	19.08
	4-8-3	125.8160	12.8057	9.82	6.5187	19.30
Cancer-Int	9-2-2	55.3120	5.6813	9.74	3.2540	17.00
	9-5-2	119.2567	12.1877	9.79	6.1856	19.28
	9-8-2	182.7007	18.7480	9.75	9.9090	18.44
Glass	9-2-2	16.9503	1.7847	9.50	1.0683	15.87
	9-5-2	37.0583	3.7550	9.87	1.9017	19.49
	9-8-2	56.2490	5.7587	9.77	3.2493	17.31
Iris	4-2-3	9.8160	1.0560	9.30	0.5247	18.71
	4-5-3	20.7807	2.1387	9.72	1.1007	18.88
	4-8-3	30.0313	3.0483	9.85	1.5717	19.11
Wine	13-2-3	19.3147	1.9877	9.72	1.0040	19.24
	13-5-3	43.6847	4.4977	9.71	2.4097	18.13
	13-8-3	65.0247	6.6180	9.83	3.3987	19.13

Tablo 5.46 ve Tablo 5.47'deki sonuçlar incelendiğinde, çalışma zamanı ve çözüm kalitesi açısından farklı YSA yapıları ile gerçekleştirilen farklı sınıflandırma problemlerinin tamamına yakınında PABC algoritması daha iyi performans göstermiştir. Paralel mimarilerin avantajlarından faydalanılarak neredeyse doğrusal hızlanma elde edilmiştir.

Tablo 5.47. ABC ve PABC algoritmaları ile eğitilen farklı YSA yapılarının ortalama sınıflandırma hata değerleri

		ABC			PABC					
		1 CPU			10 CPU			20 CPU		
Problem	YSA Yapısı	CEP_{Test}	CEP_{Egitim}	MSE	CEP_{Test}	CEP_{Egitim}	MSE	CEP_{Test}	CEP_{Egitim}	MSE
Balance	4-2-3	11.11	11.33	1.53E-01	10.92	10.98	1.48E-01	**9.96**	10.02	1.40E-01
	4-5-3	10.21	9.96	1.42E-01	8.95	9.28	1.33E-01	**8.40**	7.65	1.10E-01
	4-8-3	10.23	9.57	1.38E-01	8.61	8.91	1.24E-01	**7.54**	6.92	1.06E-01
Cancer-Int	9-2-2	3.77	2.03	3.81E-02	**2.53**	2.37	4.25E-02	3.05	3.07	4.97E-02
	9-5-2	3.81	1.90	3.56E-02	**1.92**	1.92	3.55E-02	2.90	2.80	4.78E-02
	9-8-2	4.09	1.83	3.47E-02	**1.79**	2.24	3.88E-02	1.87	1.18	2.48E-02
Glass	9-2-2	6.35	2.63	5.01E-02	4.03	5.01	6.48E-02	**2.38**	2.30	4.58E-02
	9-5-2	5.79	2.03	3.91E-02	4.65	4.61	5.82E-02	**2.45**	1.66	3.32E-02
	9-8-2	5.41	2.01	3.75E-02	2.58	2.15	4.15E-02	**1.76**	1.84	3.10E-02
Iris	4-2-3	3.77	2.92	5.55E-02	2.89	2.85	5.15E-02	**1.58**	2.35	3.51E-02
	4-5-3	2.98	2.38	4.21E-02	2.46	2.71	3.68E-02	**2.10**	1.85	3.03E-02
	4-8-3	3.42	2.08	3.65E-02	1.75	1.76	3.47E-02	**2.11**	1.49	2.53E-02
Wine	13-2-3	8.30	5.01	9.99E-02	5.04	3.78	7.08E-02	**2.44**	1.85	3.85E-02
	13-5-3	7.26	3.93	8.29E-02	4.66	4.34	7.29E-02	**2.30**	1.85	3.71E-02
	13-8-3	7.85	2.81	6.88E-02	2.89	2.13	4.01E-02	**0.96**	0.43	1.44E-02

5.9. Gezgin Satıcı Problemi İçin Paralel ABC Algoritması

Tez kapsamında şimdiye kadar yapılan uygulamaların tamamı sürekli optimizasyon problemleri üzerinedir. Bu uygulamada ise paralel sezgisel algoritmalar ile ayrık optimizasyon problemlerinin incelenmesi amaçlanmıştır. Ayrık optimizasyon problemlerinde genel olarak ayrık niceliklerin optimal olarak düzenlenmesi, gruplanması, sıralanması ve seçilmesi amaçlanmaktadır.

Gezgin satıcı problemi (Travelling Salesman Problem, TSP), ayrık problem türlerinden bir tanesidir ve "görülecek şehirler ve şehirlerarasındaki mesafe verildiğinde, her bir şehri yalnız bir defa ziyaret edecek şekilde en kısa tur mesafesini bulmak" şeklinde tanımlanmaktadır. TSP tanımlanması çok basit, ancak çözümü çok zor olduğu için ayrık problemlerin en önemlilerindendir.

Sezgisel algoritmaları ile yapılan çalışmalarda, algoritmaların ayrık modellerinin oluşturulması gerekmektedir. Algoritmaların ayrık modeller oluşturulurken, yeni çözüm üretme mekanizmaları kendi karakteristik özelliklerine göre değiştirilmektedir. Bunun yanında daha etkin yeni çözüm üretme mekanizmaları algoritmalara entegre edilerek daha başarılı sonuçlarda alınabilmektedir. Ayrıca TSP'deki şehir sayısının artması, problem boyutunu üstel olarak büyütmekte ve çözüm için gereken süreler kabul edilebilir süreleri fazlaca aşmaktadır. Bu nedenle, problemlerin çözümünde paralel hesaplama mimarilerinin avantajlarını birleştiren yapıların kullanılması oldukça önem arz etmektedir.

Bu uygulamada ABC algoritmasının ayrık modeli (Combinatorial Artificial Bee Colony, CABC) oluşturulmuş ve algoritmaya etkin komşu üretme mekanizmaları entegre edilmiştir. Ayrıca CABC algoritmasının coarse-grained paralel modeli gerçekleştirilmiştir (Parallel Combinatorial Artificial Bee Colony, PCABC). Böylece hem performans hem de çalışma zamanı bakımından başarılı sonuçlar alınması hedeflenmektedir.

PCABC algoritmasına entegre edilen etkin komşu üretme mekanizması kısa bir süre önce Albayrak ve Allahverdi tarafından geliştirilmiştir [189]. Bu mekanizma

Karaboğa ve Görkemli tarafından CABC algoritmasının seri modeline uyarlanmış ve bazı gezgin satıcı problemlerinde başarılı sonuçlar elde edilmiştir [190]. Bu etkin komşu üretme mekanizması CABC algoritmasına entegre edilirken algoritmanın karakteristiğine uygun olacak şekilde popülasyondaki bireylerin birbirlerinden faydalanmaları amaçlanmıştır. PABC algoritması da bu etkin komşu üretme mekanizması ile gerçekleştirilmiştir. Algoritmaya uyarlanan etkin komşu üretme mekanizmasının işlem adımları aşağıda verilmiştir.

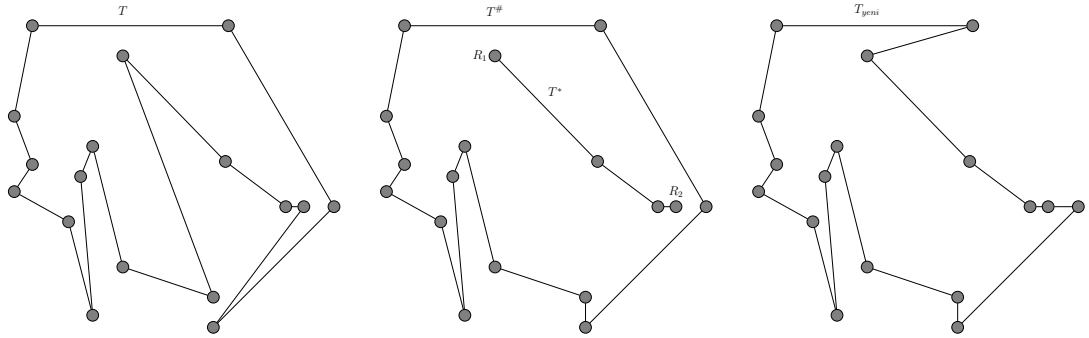
Algorithm 12: Komşu üretme mekanizması için sözde kod

Rastgele bir komşu çözüm x_k seçilmesi;
Rastgele bir şehir $x_i[j]$ seçilmesi;
Rastgele araştırma yönü parametresi ϕ belirlenmesi ($\phi = \{-1, 1\}$);
if ($\phi = -1$) **then**
    $x_k[j]$'den önce ziyaret edilen şehri $x_i[j]$'nin önüne ekle;
else
    $x_k[j]$'den sonra ziyaret edilen şehri $x_i[j]$'nin sonuna ekle;
end if{Bu işlem sonrası $T^{\#}$ kapalı turu ve T^{*} alt turu elde edilir.}
T^{*} alt turunun başlangıç şehri R_1, son şehri R_2 olarak tanımlanır;
if ($rand() \leq P_{RC}$) **then**
    T^{*} alt turunu $T^{\#}$ kapalı turunda en uygun aralığa yerleştir.;
else
    if ($rand() \leq P_{CP}$) **then**
        T^{*} alt turunu R_1 noktasından başlayarak, P_L olasılığına göre karıştırarak ekle;
    else
        R_1 ve R_2 noktaları için komşu listelerinden rastgele birer komşu seç (NL_{R1} ve NL_{R2});
        R_1 ile NL_{R1} ve R_2 ile NL_{R2} noktalarını bağla. En fazla kazanç sağlayan bağlantıyı seç;
        Gelişim gerçekleşmemiş ise tekrarla;
    end if
end if

Burada $x_i[j]$, $i$inci çözüm vektörünün $j$inci şehrini temsil etmektedir. rand() (0,1) aralığında rastgele sayı üreteci, T orjinal tur vektörü, R_1 ve R_2, T^{*} alt tur ve $T^{\#}$ kapalı tur vektörünün oluşması için belirlenen iki şehir, P_{RC} yeniden bağlanma olasılığı, P_{CP} düzeltme ve bozulma olasılığı, P_L doğrusallık olasılığı, NL_{MAX} komşu listesi boyutunu, NL_{R1} ve NL_{R2} de sırası ile R_1 ve R_2 noktalarının yakın komşu listelerinden rastgele seçilmiş birer komşularını göstermektedir.

Karmaşık fakat oldukca etkin olan bu yeni çözüm üretme mekanizmasında R_1 ve

R_2, noktaları belirlenirlen popülasyondaki rastgele seçilmiş bir başka bireyin yine rastgele seçilmiş araştırma yönüne dikkate alınarak birbirlerinden faydalanmaları sağlanır. R_1 ve R_2 noktaları belirlendikten sonra, R_1 ve R_2 aralığı T^* alt tur vektörünü, geriye kalanlar $T^\#$ kapalı tur vektörünü oluşturur. (0,1) aralığında rastgele üretilen değer P_{RC} yeniden bağlanma olasılığı parametre değerinden küçükse, T^* alt turu, $T^\#$ kapalı tur vektöründeki tüm aralıklara sırası ile denenerek, kazancın en fazla olduğu aralığa yerleştirilir ve yeni çözüm üretilmiş olur. Şekil 5.21'de T orjinal tur vektörü, belirlenen R_1 ve R_2 şehirleri ile oluşan T^* alt tur vektörü, $T^\#$ kapalı tur vektörü ve T^*'ın en uygun yere yerleştikten sonra oluşan yeni çözüm vektörü gösterilmiştir.

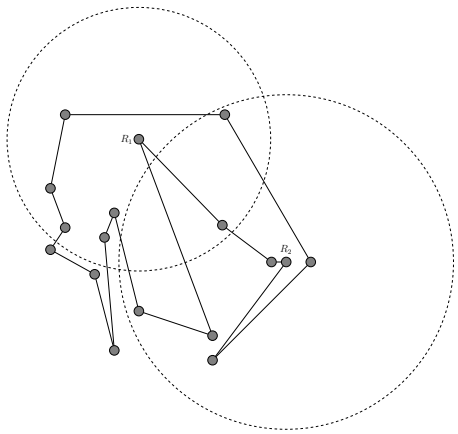


(a) (b) (c)

Şekil 5.21. Açgözlü yeniden bağlanma, a)T orjinal tur vektörü, b)T^* alt tur ve $T^\#$ kapalı tur vektörleri ve c)T_{yeni} yeni çözüm vektörü

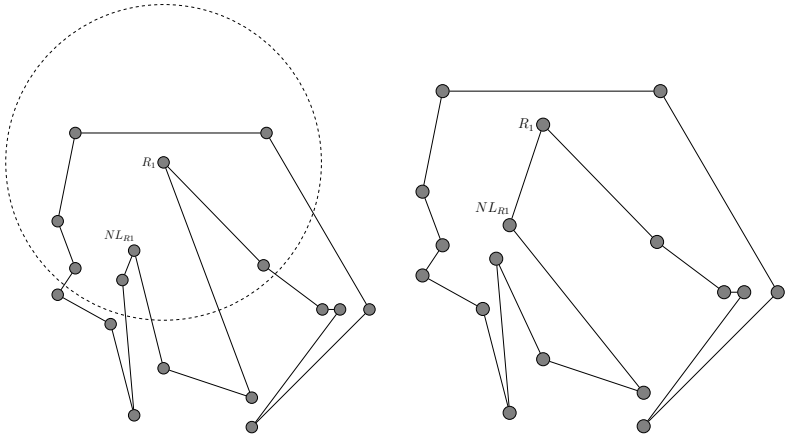
(0,1) aralığında rastgele üretilen değer P_{RC} yeniden bağlanma olasılığı parametre değerinden küçük değilse, P_{CP} düzeltme ve bozulma olasılığı değeri yine (0,1) aralığında rastgele üretilen bir değerle karşılaştırılır. Bu değer küçükse, R_1 ve R_2 şehirleri arasındaki T^* alt turu karıştırılarak tekrar yerine konulur, değilse, oluşturulan en yakın komşu listelerinden belirlenen maksimum komşu listesi boyutunu (NL_{MAX}) aşmayacak şekilde R_1 ve R_2 şehirleri için rastgele komşular seçilir (NL_{R1} ve NL_{R2}). Şekil 5.22'de R_1 ve R_2 şehirleri için $NL_{MAX} = 7$ olacak şekilde seçilebilecek en yakın komşuları grafiksel olarak gösterilmiştir.

R_1 ile NL_{R1} ve R_2 ile NL_{R2} noktaları bağlanarak varsa kazançlar hesaplanır, kazancın fazla olduğu bağlantı seçilerek yeni çözüm vektörü oluşturulur. Kazanç olmazsa en yakın komşu listelerinden tekrar başka komşular seçilerek işlemler tekrarlanır. Şekil 5.23 ve Şekil 5.24'de sırası ile R_1 ve R_2 noktalarının NL_{R1} ve

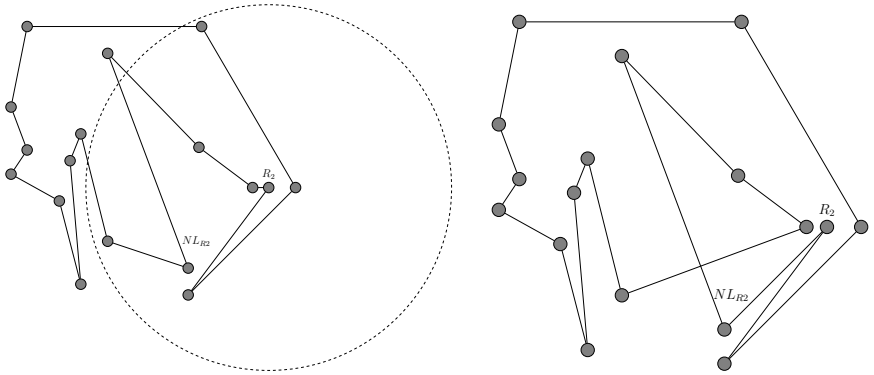


Şekil 5.22. R_1 ve R_2 şehirlerinin en yakın komşuları ($NL_{MAX} = 7$)

NL_{R2} noktaları ile bağlanmaları sonucu oluşan yeni tur vektörleri grafiksel olarak gösterilmiştir.



Şekil 5.23. R_1 ile NL_{R1} şehirleri bağlantısı sonrası oluşan tur vektörü



Şekil 5.24. R_2 ile NL_{R2} şehirleri bağlantısı sonrası oluşan tur vektörü

CABC ve PCABC algoritmalarının, entegre edilen bu yeni çözüm üretme operatörü ile performanslarını test etmek amacı ile literatürdeki Eil51, Berlin52, KroB150,

KroA200 ve Pr1002 gezgin satıcı problemleri çözümleri gerçekleştirilmiştir. Bu problemlere ait şehir sayıları ve optimum tur uzunluk bilgileri Tablo 5.48'de verilmiştir. CABC ve PCABC algoritmaları için, popülasyon büyüklüğü, limit ve toplam değerlendirme sayısı sırası ile 96, 1000 ve 20000 olarak seçilmiştir. Ayrıca algoritmalara entegre edilen etkin komşu üretme mekanizmasının parametreleri: yeniden bağlanma olasılığı, düzeltme ve bozulma olasılığı, doğrusallık olasılığı, minimum alt tur uzunluğu, maksimum alt tur uzunluğu ve komşu listesi boyutu sırası ile 0.5, 0.8, 0.2, 2, $Int(sqrt(sehir\ sayısı))$ ve 5 olarak alınmıştır. Bu değerler Albayrak ve Allahverdi'nin çalışmalarında kullandıkları değerlerdir [189]. Çözülen TSP problemleri için 30 farklı koşma sonucu elde edilen ortalama tur uzunluğu, en kısa tur uzunluğu, hata değerleri ve çalışma zamanı değerleri Tablo 5.49'da verilmiştir. Tablolarda verilen hata yüzdeleri Eşitlik (5.10) ile hesaplanmıştır.

$$Hata\% = 100x\frac{(algoritma\ ile\ bulunan\ tur) - (optimum\ tur)}{optimum\ tur} \tag{5.10}$$

Tablo 5.48. Deneysel çalışmalarda kullanılan TSP problemlerinin özellikleri

Problem	Şehir Sayısı	Optimum Tur Uzunluğu
Eil51	51	426
Berlin52	52	7542
KroB150	150	26130
KroA200	200	29368
Pr1002	1002	259045

Tablo 5.49'deki sonuçlar incelendiğinde CABC ve PCABC algoritmaları ile elde edilen sonuçlar birbirine yakın olmakla beraber her ikisi ile de oldukça başarılı sonuçlar elde edilmiştir. Çalışma zamanları kıyaslandığında PCABC algoritmasının daha kısa sürede çözüm sağladığı görülmektedir. Bunun yanında şehir sayısının az olduğu problemlerde şehir sayısının fazla olduğu problemlere oranla daha fazla zamansal kazanç sağlandığı gözlemlenmiştir. Özellikle şehir sayısının çok fazla olduğu problemlerde algoritmaya entegre edilen yeni çözüm üretme operatörü her bir alt popülasyonda çok farklı sürelerde çalışbilmektedir. Dolayısı ile bu alt popülasyonların iletişim için birbirleri ile seknronizasyon gerekliliği zamansal kazancı azaltmıştır.

Tablo 5.49. CABC ve PCABC algoritmaları ile TSP problemleri çözüm hata değerleri

Problem	Algoritma	CPU Sayısı	En iyi tur	Ort. tur	En iyi hata	Ort. hata	Zaman
Eil51	CABC	1	427	427	0.23	0.23	9.3723
	PCABC	2	427	427	0.23	0.23	5.0127
		4	427	427	0.23	0.23	2.9140
		8	426	427	0.00	0.23	1.8710
Berlin52	CABC	1	7542	7542	0.00	0.00	8.8923
	PCABC	2	7542	7542	0.00	0.00	5.0020
		4	7542	7542	0.00	0.00	2.9223
		8	7542	7542	0.00	0.00	1.8783
KroB150	CABC	1	26207	26432	0.29	1.15	25.5127
	PCABC	2	26141	26396	0.04	1.01	15.8947
		4	26141	26464	0.04	1.27	11.0639
		8	26251	26462	0.46	1.27	8.6947
KroA200	CABC	1	29455	29572	0.29	0.69	40.4843
	PCABC	2	29436	29589	0.23	0.75	26.7193
		4	29455	29594	0.29	0.76	19.7290
		8	29541	29632	0.58	0.89	16.2870
Pr1002	CABC	1	266226	269151	2.77	3.90	230.1323
	PCABC	2	265906	269090	2.64	3.87	198.2139
		4	265492	269032	2.49	3.86	173.1399
		8	265908	269149	2.65	3.90	188.3500

Gerçekleştirilen yaklaşımın performansını değerlendirmek amacıyla CABC algoritması ile elde edilen sonuçlar, Berlin52, KroB150 ve KroA200 problemleri için Albayrak ve Allahverdi'nin genetik algoritmanın farklı yeni çözüm üretme metotları ve kendi geliştirdikleri yeni çözüm üretme metodunu kullanarak buldukları sonuçlarla karşılaştırılmıştır [189]. Sonuçlar Tablo 5.50'de verilmiştir.

Tablo 5.50. PCABC ve farklı GA modelleri ile TSP problemleri çözüm hata değerleri

Metot	Berlin52		KroB150		KroA200	
	En iyi hata	Ort. hata	En iyi hata	Ort. hata	En iyi hata	Ort. hata
EXC	0.0000	2.0353	2.9277	5.0919	2.5061	4.9877
DISP	0.0663	1.4240	7.8263	10.4635	6.7284	8.8276
INV	0.0000	1.1854	7.1183	9.0521	6.2245	8.3594
INS	0.0000	0.1525	1.8178	4.5488	2.1554	4.2819
SIM	0.0000	0.6099	2.8397	3.8596	1.5766	3.2457
SCM	0.0000	0.0000	3.2836	7.0609	3.9873	5.4502
GSM	0.0000	0.9931	3.4520	4.9541	4.4675	5.7018
GSTM	0.0000	0.0000	0.9644	1.7616	0.8683	1.5432
PCABC	0.0000	0.0000	0.0400	1.0100	0.2300	0.7500

Tablo 5.50'deki sonuçlar incelendiğinde Berlin52 probleminin nispeten kolay olması nedeniyle genetik algoritmanın farklı yeni çözüm üretme metotları ve PCABC algoritması ile oldukça başarılı sonuçlar elde edildiği söylenebilir. Bunun yanında nispeten daha zor problemler olan KroB150 ve KroA200 çözümlerinde PCABC algoritmasının genetik algoritmanın tüm metotlarından daha iyi sonuç verdiği görülmektedir.

6. BÖLÜM

TARTIŞMA - SONUÇ VE ÖNERİLER

6.1. Sonuçlar

Gelişen teknoloji; biyoloji, tıp, beşeri bilimler ve yönetim bilimleri gibi birçok alanda daha karmaşık sistemlerin ortaya çıkmasına sebep olmaktadır. Literatürdeki geleneksel hesaplama tekniklerinin belirleyici ve tam sonuç verme gibi avantajlarına rağmen, bu teknikler ile karmaşık sistemlerin çözümleri genellikle yetersiz ya da mümkün olamamaktadır. Bu nedenle genel amaçlı, makul çözümler üreten etkili çözüm yöntemleri geliştirilmektedir. Ayrıca geliştirilen bu yöntemlerin çok işlemcili donanım mimarileri üzerinde çalışabilecek şekilde uyarlanması ile de çözüm kalitelerinin iyileştirilmesi ve çözüm sürelerinin kısaltılması amaçlanmaktadır. Bu amaçlarla bu tez çalışmasında literatürdeki mevcut güncel bazı popülasyon tabanlı sezgisel algoritmaların seri ve paralel modelleri gerçekleştirilmiş, başarımları karşılaştırılmış, bazı iyileştirme stratejileri önerilmiş ve paralel modellerin ihtiyaç duyduğu ek parametrelerin analizleri yapılmıştır. Bunun yanında, farklı algoritmaların veya çözüm stratejilerinin dezavantajlarını ortadan kaldırmak amacı ile hbrid modeller oluşturulmuştur. Gerçekleştirilen modellerin performans analizi, basit fonksiyonlar, basit fonksiyonların ötelenmesi, döndürülmesi ve toplanması ile elde edilen, büyük boyutlu oldukça zor test fonksiyonları ve yapay sinir ağları eğitimi üzerinde yapılmıştır. Ayrıca ayrık problemlerin çözümü için bazı algoritmaların yeni ayrık paralel modelleri önerilmiş ve performansları literatürdeki bazı çalışmalarla karşılaştırılmıştır.

Tez çalışmasının birinci bölümünde, esnek hesaplama yöntemlerini oluşturan başlıca bileşenler kısaca tanıtılmış, esnek hesaplama yöntemleri ile geleneksel

hesaplama tekniklerinin karşılaştırılması yapılmış ve gerçekleştirilen uygulamalarla ilgili literatür taraması verilmiştir.

Çalışmanın ikinci bölümünde, seri hesaplama sistemleri ve bilgisayar donanımları kısaca anlatılmıştır. Paralel bilgisayar sistemleri, paralel algoritmalar, paralel programlama ve temel uygulamaları hakkında bilgiler verilmiştir. Üçüncü bölümde esnek hesaplama yöntemlerinden yapay sinir ağları ve tez kapsamında gerçekleştirilen popülasyon tabanlı sezgisel algoritmalar tanıtılmıştır. Dördüncü bölümde algoritmaların paralelleştirilme modelleri verilmiştir.

Beşinci bölümde gerçekleştirilen uygulamalar ve elde edilen sonuçlar verilmiştir. Bu bölümde ABC algoritması için temel asenkron modeli ile önerilen senkron modeli performans açısından karşılaştırılmış ve senkron ABC algoritmasının master-slave parallelleştirme modeli için çalışma zamanı değerleri verilmiştir. Senkron ABC ve asenkron ABC algoritmalarının performansları birbirine benzer çıkarken, paralel gerçekleştirimde sınırlı hızlanma elde edilebilmiştir. PSO algoritmasının performansını iyileştirmek amacı ile bazı stratejiler önerilmiştir. Geliştirilmiş PSO algoritmasının fine-grained parallelleştirme modeli için performans değerleri verilmiştir. Yine bu bölümde ABC algoritmasının coarse-grained parallelleştirme modeli gerçekleştirilerek, bu modele özgü kontrol parametrelerinin performans üzerindeki etkisi incelenmiş ve bu parametreler için tavsiye edilebilecek değerler belirlenmiştir. Önerilen model YSA eğitiminde kullanılmış ve farklı sınıflandırma problemlerinin çözümleri gerçekleştirilmiştir. Hybrid paralelleştime modelini gerçekleştirmek amacı ile de literatürde yeni tanımlanan TLBO algoritması tercih edilmiştir.

Ayrıca literatürde oldukça popüler veya yeni olan bazı algoritmaların (ABC, CS, DE, FF, GS, PSO ve TLBO) seri ve paralel modellerinin karşılaştırıldığı detaylı bir çalışma yapılmıştır. Sonuçlardan ABC ve DE algoritmasının çalışma zamanının ve performansının çalışmada kıyaslanan diğer algoritmalardan genel olarak daha iyi olduğu gözlemlenmiştir.

Bu bölümde yine DE algoritmasının farklı stratejileri ile oluşturulan paralel portföy yapısı gerçekleştirilmiştir. Sonuçlardan oluşturulan portföy yapısının zamansal

kazanç sağlamanın yanı sıra kendisini oluşturan stratejilere oranla daha kararlı sonuçlar ürettiği gözlemlenmiştir.

Yine bu bölümde ayrık PABC algoritması gerçekleştirilmiş, algoritmaya etkin komşu üretme mekanizmaları entegre edilmiş ve önerilen model ile ayrık bir problem türü olan gezgin satıcı problemlerinin çözümü gerçekleştirilmiştir. Sonuçlardan paralel hızlanmanın problemlerdeki şehir sayısına bağlı olarak değiştiği gözlemlenirken, çözüm kalitesi olarak ayrık PABC algoritması ile oldukça başarılı sonuçlar elde edilmiştir. Özellikle şehir sayısının fazla olduğu nispeten daha zor problemlerde ayrık PABC algoritması karşılaştırıldığı metotlardan çok daha iyi sonuçlar vermiştir.

Özetle gerçekleştirilen uygulamalardan, algoritmalar için önerilen iyileştirme stratejileri ile çözüm kaliteleri arttırılmış, paralel hesaplama sistemlerinin kullanımı ile çözüm süreleri önemli oranda kısaltılmıştır. Elde edilen sonuçlardan, paralel hesaplama sistemleri üzerinde çalışan paralel algoritmalar ile karmaşık sistemlerin etkin bir şekilde çözülebileceği görülmüştür.

6.2. Gelecekte Yapılacak Çalışmalar

Çözüm kaliteleri arttırılan ve paralel gerçekleştirimleri yapılan algoritmaların farklı gerçek dünya problemleri çözümlerinde kullanılması gelecekte yapılacak çalışmalar olarak düşünülmektedir. Ayrıca paralel programlamanın zorluğu ve çok işlemcili donanım mimarilerinin kurulum ve bakım maliyetlerinin yüksek olması sebebi ile önerilen ve geliştirilen algoritmaların daha düşük maliyetli yüksek hesaplama kapasitesine sahip ekran kartları gibi farklı donanımlar üzerinde çalışabilecek şekilde uyarlanmaları planlanmaktadır.

KAYNAKLAR

1. Holland, J. H., 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI, 217 pp.

2. Glover, F., 1989. Tabu search - part i. **ORSA Journal on Computing**, **1**(3):190–206.

3. Glover, F., 1990. Tabu search - part ii. **ORSA Journal on Computing**, **2**(1):4–32.

4. Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P., 1983. Optimization by simulated annealing. **Science**, **220**(4598):671–680.

5. Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E., 1953. Equations of state calculations by fast computing machines. **Journal of Chemical Physics**, **21**(6):1087–1092.

6. Dorigo, M., Maniezzo, V., and Colorni, A., 1991. Positive feedback as a search strategy. Technical Report 91–016, Politecnico di Milano, Italy.

7. Karaboga, D., 2005. An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department.

8. Price, K., Storn, R., and Lampinen, J., 2005. Differential Evolution: A Practical Approach to Global Optimization. Springer–Verlag, Berlin, Germany, pp 292.

9. Kennedy, J. and Eberhart, R., 1995. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, pp. 1942–1948. WA, Australia.

10. Yang, X. S. and Deb, S., 2009. Cuckoo search via lévy flights. In *World Congress on Nature and Biologically Inspired Computing (NaBIC 2009)*, pp. 210–214. IEEE.

11. Yang, X. S., 2008. Nature-Inspired Metaheuristic Algorithms. Luniver Press, UK, 147 pp. ISBN 1-905986-10-6.

12. Rashedi, E., Nezamabadi-pour, H., and Saryazdi, S., 2009. Gsa: A gravitational search algorithm. **Information Sciences**, **179**(13):2232–2248.

13. Rao, R. V., Savsani, V. J., and Vakharia, D. P., 2011. Teaching-learning-based optimization: A novel method for constrained mechanical design optimization problems. **Computer-Aided Design**, **43**:303–315.

14. Rao, R. V., Savsani, V. J., and Vakharia, D. P., 2012. Teaching-learning-based optimization: An optimization method for continuous non-linear large scale problems. **Information Sciences**, **183**:1–15.

15. Quan, H. and Shion, X., 2008. On the analysis of performance of the improved artificial-bee-colony algorithm. In *Fourth International Conference on Natural Computation (ICNC'08)*. Jinan, China.

16. Tsai, P. W., Pan, J. S., Liao, B. Y., and Chu, S. C., 2009. Enhanced artificial bee colony optimization. **International Journal of Innovative Computing, Information and Control**, **5**(12):1–12.

17. Kang, F., Li, J., and Xu, Q., 2009. Structural inverse analysis by hybrid simplex artificial bee colony algorithms. **Computers and Structures**, **87**(13–14):861–870.

18. Karaboga, D. and Akay, B., 2011. A modified artificial bee colony (ABC) algorithm for constrained optimization problems. **Applied Soft Computing**, **11**:3021–3031.

19. Pana, Q. K., Tasgetiren, M. F., Suganthan, P. N., and Chuad, T. J., 2011.

A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. **Information Sciences**, **181**(12):2455–2468.

20. Akay, B. and Karaboga, D., 2012. A modified artificial bee colony algorithm for real-parameter optimization. **Information Sciences**, **192**:120–142.

21. Manoj, V. J. and Elias, E., 2012. Artificial bee colony algorithm for the design of multiplier-less nonuniform filter bank transmultiplexer. **Information Sciences**, **192**:193–203.

22. Yildiz, A. R., 2013. A new hybrid artificial bee colony-based approach for optimization of multi-pass turning operations. **Information Sciences**, **220**:399–407.

23. Zhang, J. and Sanderson, A. C., 2009. Jade: Adaptive differential evolution with optional external archive. **IEEE Transactions On Evolutionary Computation**, **13**(5):945–958.

24. Chong, C. K., Mohamad, M. S., Deris, S., Shamsir, M. S., Choon, Y. W., and L., E.-C., 2012. Improved differential evolution algorithm for parameter estimation to improve the production of biochemical pathway. **International Journal of Interactive Multimedia and Artificial Intelligence**, **1**(5):22–29.

25. He, Q. and Han, C., 2006. An improved particle swarm optimization algorithm with disturbance term. **Computational Intelligence and Bioinformatics**, **4115**:100–108.

26. Liu, Y., Wang, G., Chen, H., Dong, H., Zhu, X., and Wang, S., 2011. An improved particle swarm optimization for feature selection. **Journal of Bionic Engineering**, **8**(2):191–200.

27. Tsai, H. C., Tyan, Y. Y., Wu, Y. W., and Lin, Y. H., 2012. Isolated particle swarm pptimization with particle migration and global best adoption. **Engineering Optimization**, **44**(12):1405–1424.

28. Rao, R. V. and Patel, V., 2013. An improved teaching-learning-based optimization algorithm for solving unconstrained optimization problems. **Scientia Iranica**, **20**(3):710–720.

29. Rao, R. V. and Patel, V., 2012. An elitist teaching-learning-based optimization algorithm for solving complex constrained optimization problem. **International Journal of Industrial Engineering Computation**, **3**:535–560.

30. Corne, D., Dorigo, M., and Glover, F., 1999. New Ideas in Optimization. McGraw Hill, 493 pp.

31. Gamperle, R., Muller, S., and Koumoutsakos, P., 2002. A parameter study for differential evolution. In *WSEAS International Conference on Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, pp. 293–298.

32. Vesterstrom, J. and Thomsen, R., 2004. A comparative study of differential evolution particle swarm optimization and evolutionary algorithms on numerical benchmark problems. In *IEEE Congress on Evolutionary Computation (CEC2004)*, volume 3, pp. 1980–1987. Piscataway, New Jersey.

33. Mezura-Montes, E., Velázquez-Reyes, J., and Coello Coello, C. A., 2006. A comparative study of differential evolution variants for global optimization. In *Genetic and Evolutionary Computation*, pp. 485–492. Seattle, WA.

34. Karaboğa, D. and Akay, B., 2008. On the performance of artificial bee colony (ABC) algorithm. **Applied Soft Computing Journal**, **8**:687–697.

35. Abbass, H. A., Sarkar, R., and Newton, C., 2001. A pareto differential evolution approach to vector optimisation problems. In *IEEE Congress on Evolutionary Computation*, pp. 971–978. Seoul, Korea.

36. Abbass, H., 2002. The self-adaptive pareto differential evolution algorithm. In *IEEE Congress on Evolutionary Computation*, pp. 831–836. Honolulu, HI.

37. Zaharie, D., 2002. Critical values for the control parameters of differential evolution algorithms. In *8th International Conference on Soft Computing*, pp. 62–67.

38. Zaharie, D., 2003. Control of population diversity and adaptation in differential evolution algorithms. In R. Matousek and P. Osmera, editors, *9th International Conference on Soft Computing*, pp. 41–46. Czech Republic.

39. Zaharie, D. and Petcu, D., 2003. Adaptive pareto differential evolution and its parallelization. In *5th International Conference Parallel Process. Appl. Math.*, pp. 261–268. Czestochowa, Poland.

40. Liu, J. and Lampinen, J., 2005. A fuzzy adaptive differential evolution algorithm. **Soft Computing**, **9**(6):448–462.

41. Xue, F., Sanderson, A. C., Bonissone, P. P., and Graves, R. J., 2005. Fuzzy logic controlled multiobjective differential evolution. In *IEEE International Conference on Fuzzy Systems*, pp. 720–725. HReno, NV.

42. Qin, A. K. and Suganthan, P. N., 2005. Self-adaptive differential evolution algorithm for numerical optimization. In *IEEE Congress on Evolutionary Computation*, volume 2, pp. 1785–1791.

43. Huang, V. L., Qin, A. K., and Suganthan, P. N., 2006. Self-adaptive differential evolution algorithm for constrained real-parameter optimization. In *IEEE Congress on Evolutionary Computation*, pp. 17–24.

44. Brest, J., Greiner, S., Boskovic, B., Mernik, M., and Zumer, V., 2006. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. **IEEE Transactions on Evolutionary Computation**, **2**:82–102.

45. Teo, J., 2006. Exploring dynamic self-adaptive populations in differential evolution. **Soft Computing**, **10**(8):673–686.

46. Qin, A. K., Huang, V. L., and Suganthan, P. N., 2009. Differential evolution

algorithm with strategy adaptation for global numerical optimization. **IEEE Transactions on Evolutionary Computation**, **13**(2):398–417.

47. Zhan, Z. H. and Zhang, J., 2009. Parallel Particle Swarm Optimization with Adaptive Asynchronous Migration Strategy, chapter Algorithms and Architectures for Parallel Processing Lecture Notes in Computer Science, pp. 490–501. Springer Berlin Heidelberg.

48. Peng, F., Tang, K., Chen, G., and Yao, X., 2010. Population-based algorithm portfolios for numerical optimization. **IEEE Transactions on Evolutionary Computation**, **14**(5):782–800.

49. Mallipeddi, R. and Suganthan, P. N., 2010. Differential evolution algorithm with ensemble of parameters and mutation and crossover strategies. In Swarm, Evolutionary, and Memetic Computing, pp. 71–78. Springer Berlin Heidelberg.

50. Mallipeddi, R., Suganthan, P. N., Pan, Q. K., and Tasgetiren, M. F., 2011. Differential evolution algorithm with ensemble of parameters and mutation strategies. **Applied Soft Computing**, **11**(2):1679–1696.

51. Alam, M. S. and Islam, M. M., 2011. Artificial bee colony algorithm with self-adaptive mutation: A novel approach for numeric optimization. In *TENCON 2011, IEEE Region 10 Conference*, pp. 49–53. Bali.

52. Huang, Z. and Chen, Y., 2013. An improved differential evolution algorithm based on adaptive parameter. **Journal of Control Science and Engineering**, **2013**:1–5.

53. Wang, H., Wang, B., and Wuc, Z., 2013. Particle swarm optimization with adaptive mutation for multimodal optimization. **Applied Mathematics and Computation**, **221**:296–305.

54. Yu, W. J., Zhang, J., and Chen, W. N., 2013. Adaptive artificial bee colony optimization. In *Genetic and Evolutionary Computation Conference*, pp. 153–158. New York, USA.

55. Poli, R., 2007. An analysis of publications on particle swarm optimisation applications. Technical report csm-469, Department of Computer Science, University of Essex, UK.

56. Hruschka, E. R., Campello, R. J. G. B., Freitas, A. A., and de Carvalho, A. C. P. L. F., 2009. A survey of evolutionary algorithms for clustering. **IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews**, **39**(2):133–155.

57. Karaboga, D. and Akay, B., 2009. A survey: Algorithms simulating bee swarm intelligence. **Artificial Intelligence Review**, **31**(1):55–68.

58. Neri, F. and Tirronen, V., 2010. Recent advances in differential evolution:a survey and experimental analysis. **Artificial Intelligence Review**, **33**(1–2):61–106.

59. Das, S. and Suganthan, P. N., 2011. Differential evolution: A survey of the state-of-the-art. **IEEE Transactions on Evolutionary Computation**, **15**(1):4–31.

60. Pedemonte, M., Nesmachnow, S., and Cancela, H., 2011. A survey on parallel ant colony optimization. **Applied Soft Computing**, **11**(8):5181–5197.

61. Manda, K., Satapathy, S. C., and Poornasatyanarayana, B., 2012. Population based meta-heuristic techniques for solving optimization problems: A selective survey. **International Journal of Emerging Technology and Advanced Engineering**, **2**(11):206–211.

62. Barros, R. C., Basgalupp, M. P., de Carvalho, A. C. P. L. F., and Freitas, A. A., 2012. A survey of evolutionary algorithms for decision-tree induction. **IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews**, **42**(3):291–312.

63. Karaboga, D., Gorkemli, B., Ozturk, C., and Karaboga, N., 2014. A comprehensive survey: Artificial bee colony (ABC) algorithm and applications. **Artificial Intelligence Review**, **42**:21–57.

64. Holland, J. H., 1959. A universal computer capable of executing an arbitrary number of sub-programs simultaneously. In *Eastern Joint Computer Conference*, pp. 108–113. New York, NY, USA.

65. Holland, J. H., 1960. Iterative circuit computers. In *Western Joint Computer Conference*, pp. 259–265. New York, NY, USA.

66. Bethke, A. D., 1976. Comparisons of genetic algorithms and gradient-based optimizers on parallel processors: Efficiency of use of processing capacity. Technical Report 197, University of Michigan.

67. Grefenstette, J. J., 1981. Parallel adaptive algorithms for function optimization. Technical Report CS–81–19, Vanderbilt University, Computer Science Department, Nashville, TN.

68. Grosso, P. B., 1985. Computer simulation of genetic adaptation: Parallel sub-component interaction in a multilocus model. Ph.D. thesis, University of Michigan, Ann Arbor.

69. Garcia, B. L., Potvin, J. Y., and Rousseau, J. M., 1995. A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints. **Computers and Operations Research**, **21**(9):1025–1033.

70. Roussel-Ragot, P. and Dreyfus, G., 1990. A problem independent parallel implementation of simulated annealing: Models and experiments. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and System**, **9**(8):827–735.

71. Rudolph, G., 1991. Global optimization by means of distributed evolution strategies. **Parallel Problem Solving from Nature**, **496**:209–213.

72. Duncan, B. S., 1993. Parallel evolutionary programming. In *2nd Annual Conference on Evolutionary Programming*, pp. 202–208. San Diego, CA.

73. Dorigo, M., 1993. Parallel ant system: an experimental study. Unpublished manuscript.

74. Fogarty, T. C. and Huang, R., 1991. Implementing the genetic algorithm on transputer based parallel processing systems. **Parallel Problem Solving from Nature**, pp. 145–149.

75. Abramson, D. and Abela, J., 1992. A parallel genetic algorithm for solving the school timetabling problem. In *Division of Information Technology, C.S.I.R.O*, pp. 1–11.

76. Abramson, D., Mills, G., and Perkins, S., 1993. Parallelisation of a genetic algorithm for the computation of efficient train schedules. In *Parallel Computing and Transputers Conference*, pp. 139–149. Brisbane.

77. Venter, G. and Sobieszczanski-Sobieski, J., 2005. A parallel particle swarm optimization algorithm accelerated by asynchronous evaluations. **Journal of Aerospace Computing, Information, and Communication**, **3**:123–137.

78. Scriven, I., Ireland, D., Lewis, A., Mostaghim, S., and Branke, J., 2008. Asynchronous multiple objective particle swarm optimisation in unreliable distributed environments. In *Evolutionary Computation, (CEC2008)*, pp. 2481–2486. Hong Kong.

79. Narasimhan, H., 2009. Parallel artificial bee colony (PABC) algorithm. In *World Congress on Nature and Biologically Inspired Computing (NaBIC 2009)*. India.

80. Basturk, A. and Akay, R., 2012. Parallel implementation of synchronous type artificial bee colony algorithm for global optimization. **Journal of Optimization Theory and Applications**, **155**(3):1095–1104.

81. Manderick, B. and Spiessens, P., 1989. Fine-grained parallel genetic algorithms. In J. Schaffer, editor, *Third International Conference on Genetic Algorithms*, pp. 428–433. San Mateo, CA.

82. Sarma, J. and Jong, K. D., 1996. An Analysisof the Effects of Neighborhoodsize and Shape on Local Selection Algorithms, chapter Parallel Problem Solving from Nature IV, pp. 236–244. Springer-Verlag, Berlin.

83. Tanese, R., 1987. Parallel genetic algorithms for a hypercube. In *2nd International Conferance on Genetic Algorithms*, pp. 177–183. Pittsburg.

84. Pettey, C. B., Leuze, M. R., and Grefenstette, J. J., 1987. A parallel genetic algorithm. In *2nd International Conferance on Genetic Algorithms*, pp. 155–161. Pittsburg.

85. Pettey, C. C., Leuze, M. R., and Grefenstette, J. J., 1987. Genetic algorithms on a hypercube multiprocessor. In *Second Conference on Hypercube Multiprocessors*, pp. 333–341.

86. Cohoon, J. P., Hegde, S. U., Martin, W. N., and Richards, D., 1987. Punctuated equilibria: A parallel genetic algorithm. In J. Grefenstette, editor, *Second International Conference on Genetic Algorithms*, pp. 148–154. Hillsdale, NJ.

87. Tanese, R., 1989. Distributed genetic algorithms. In J. D. Schaffer, editor, *Third International Conference on Genetic Algorithms*, pp. 434–439. San Mateo, CA.

88. Tanese, R., 1989. Distributed genetic algorithms for function optimization. Ph.D. thesis, University of Michigan, Ann Arbor.

89. Muhlenbein, H., Schomisch, M., and Born, J., 1991. The parallel genetic algorithm as function optimizer. **Parallel Computing**, **17**:619–632.

90. Gordon, V. S. and Whitley, D., 1993. Serial and parallel genetic algorithms as function optimizers. In S. Forrest, editor, *Fifth International Conference on Genetic Algorithms*, pp. 177–183. San Mateo, CA.

91. Chen, R. J., Meyer, R. R., and Yackel, J., 1993. A genetic algorithm for diversity minimization and its parallel implementation. In S. Forrest, editor, *Fifth International Conference on Genetic Algorithms*, pp. 163–170. San Mateo, CA.

92. Schutte, J. F., Reinbolt, J. A., Fregly, B. J., Haftka, R. T., and George, A. D.,

2004. Parallel global optimization with the particle swarm algorithm. **Int J Numer Methods Engineering**, **61**(13):2296–2315.

93. Koh, B. I., George, A. D., Haftka, R. T., and Fregly, B. J., 2006. Parallel asynchronous particle swarm optimization. **Int J Numer Methods Engineering**, **67**(4):578–595.

94. Chen, D. J., Lee, C. Y., Park, C. H., and Mendes, P., 2007. Parallelizing simulated annealing algorithms based on high-performance computer. **Journal of Global Optimization**, **39**:261–289.

95. Tasoulis, D. K., Pavlidis, N. G., Plagianakos, V. P., and Vrahatis, M. N., 2004. Parallel differential evolution. In *IEEE Congr. Evol. Comput. (CEC)*, pp. 2023–2029. Portland, OR.

96. Kwedlo, W. and Bandurski, K., 2006. A parallel differential evolution algorithm for neural network training. In *International Symposium on Parallel Computing in Electrical Engineering*, pp. 319–324. Bialystok.

97. Subotic, M., Tuba, M., and Stanarevic, N., 2010. Parallelization of the artificial bee colony (ABC) algorithm. In *11th WSEAS International Conference on Nural Networks and 11th WSEAS International Conference on Evolutionary Computing and 11th WSEAS International Conference on Fuzzy systems*, pp. 191–196.

98. Luo, R., Pan, S. T., Tsai, P. W., and Pan, J. S., 2010. Parallelized artificial bee colony with ripple-communication strategy. In *Fourth International Conference on Genetic and Evolutionary Computing*, pp. 350–353. Shenzhen.

99. Subotic, M., Tuba, M., and Stanarevic, N., 2011. Different approaches in parallelization of the artificial bee colony algorithm. **International Journal of Mathematical Models and Methods in Applied Sciences**, **5**(4):755–762.

100. Parpinelli, R. S., Benitez, C. M. V., and Lopes, H. S., 2011. Parallel Approaches

for the Artificial Bee Colony Algorithm, chapter Handbook of Swarm Intelligence, pp. 329–345. Springer-Verlag Berlin Heidelberg.

101. Chu, S. C., Roddick, J. F., and Pan, J. S., 2005. A parallel particle swarm optimization algorithm with communication strategies. **Journal of Information Science and Engineering**, **21**:809–818.

102. Chu, S. C. and Pan, J. S., 2006. Intelligent parallel particle swarm optimization algorithms. **Studies in Computational Intelligence**, **22**:159–175.

103. König, M., 2010. Design and implementation of parallel island models for CMA-ES. Master's thesis, Swiss Federal Institute of Technology Zurich.

104. Basturk, A. and Akay, R., 2013. Performance analysis of the coarse-grained parallel model of the artificial bee colony algorithm. **Information Sciences**, **253**:34–55.

105. Subotic, M., Tuba, M., Bacanin, N., and Simian, D., 2012. Parallelized cuckoo search algorithm for unconstrained optimization. In *5th WSEAS Congress on Applied Computing Conference*, pp. 151–156. Stevens Point, Wisconsin, USA.

106. Subotic, M., Tuba, M., and Stanarevic, N., 2012. Parallelization of the firefly algorithm for unconstrained optimization problems. In *Latest Advances in Information Science and Applications*, pp. 264–269.

107. Benitez, C. M. V. and Lopes, H. S., 2010. Parallel artificial bee colony algorithm approaches for protein structure prediction using the 3dhp-sc model. **Intelligent Distributed Computing**, **4**:255–264.

108. Parpinelli, R. S., Benitez, C. M. V., and Lopes, H. S., 2010. Parallel Approaches for the Artificial Bee Colony Algorithm, volume 8, chapter Handbook of Swarm Intelligence, Adaptation, Learning, and Optimization, pp. 329–345. Springer, Berlin.

109. Cahon, S., Melab, N., and Talbi, E. G., 2004. Paradiseo: A framework for

the reusable design of parallel and distributed metaheuristics. **Journal of Heuristics**, **10**:357–380.

110. Banos, R., Gil, C., Montoya, M. G., and Ortega, J., 2003. A parallel evolutionary algorithm for circuit partitioning. In *Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing*, pp. 365–371. Genova, Italy.

111. Parrilla, M., Aranda, J., and Dormido-Canto, S., 2005. Parallel Evolutionary Computation: Application of an EA to Controller Design, chapter Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach, pp. 153–162. Springer Berlin Heidelberg.

112. Chu, D. and Zomaya, A., 2006. Parallel ant colony optimization for 3d protein structure prediction using the hp lattice model. **Studies in Computational Intelligence**, **22**:177–198.

113. Melab, N., Talbi, E. G., and Cahon, S., 2006. On parallel evolutionary algorithms on the computational grid. **Studies in Computational Intelligence**, **22**:117–132.

114. Melab, N., Cahon, S., and Talbi, E. G., 2006. Grid computing for parallel bioinspired algorithms. **Journal of Parallel and Distributed Computing**, **66**:1052–1061.

115. Murugavalli, S. and Rajamani, V., 2006. A high speed parallel fuzzy c-mean algorithm for brain tumor segmentation. **BIME Journal**, **6**:29–34.

116. Manfrin, M., Birattari, M., Stützle, T., and Dorigo, M., 2006. Parallel ant colony optimization for the traveling salesman problem. Technical report, Bruxelles, Belgium: IRIDIA.

117. G., T. E., Mostaghim, S., Okabe, T., Ishibuchi, H., Rudolph, G., and Coello Coello, C. A., 2008. Parallel approaches for multiobjective optimization. **Multiobjective Optimization**, **5252**:349–372.

118. Kumar, V. and Mittal, A. P., 2010. Parallel fuzzy P + fuzzy I + fuzzy D controller: Design and performance evaluation. **International Journal of Automation and Computing**, **7**(4):463–471.

119. Zhu, W. and Curry, J., 2009. Parallel ant colony for nonlinear function optimization with graphics hardware acceleration. In *IEEE International Conference on Systems, Man and Cybernetics*, pp. 1803–1808. San Antonio, TX.

120. Zhou, Y. and Tan, Y., 2009. Gpu-based parallel particle swarm optimization. In *IEEE Congress on Evolutionary Computation (CEC2009)*, pp. 1493–1500. Trondheim.

121. Li, J., Zhang, L., and Liu, L., 2009. A parallel immune algorithm based on fine-grained model with gpu-acceleration. In *Fourth International Conference on Innovative Computing, Information and Control (ICICIC)*, pp. 683–686. Kaohsiung.

122. Zhu, W., Curry, J., and Marquez, A., 2010. Simd tabu search for the quadratic assignment problem with graphics hardware acceleration. **International Journal of Production Research**, **48**(4):1035–1047.

123. Pospichal, P., Jaros, J., and Schwarz, J., 2010. Parallel Genetic Algorithm on the CUDA Architecture, chapter Applications of Evolutionary Computation, pp. 442–451. Springer-Verlag, Berlin Heidelberg.

124. de Veronese, L. P. and Krohling, R. A., 2010. Differential evolution algorithm on the gpu with c-cuda. In *IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–7. Barcelona.

125. Zhu, W., 2011. Nonlinear optimization with a massively parallel evolution strategy-pattern search algorithm on graphics hardware. **Applied Soft Computing**, **11**(2):1770–1781.

126. Ferreiro, A. M., García, J. A., López-Salas, J. G., and Vázquez, C., 2013. An efficient implementation of parallel simulated annealing algorithm in GPUs. **Journal of Global Optimization**, **57**(3):863–890.

127. Paukste, A., 2013. Genetic Algorithm on GPU Performance Optimization Issues, chapter Intelligent Data Engineering and Automated Learning (IDEAL), pp. 529–536. Springer Berlin Heidelberg.

128. Grama, A., Gupta, A., Karypis, G., and Kumar, V., 2003. Introduction to Parallel Computing. Addison Wesley, Edinburgh, 856 pp.

129. Flynn, M., 1972. Some computer organizations and their effectiveness. **IEEE Transactions on Computers**, **21**(9):948–960.

130. `https://computing.llnl.gov/tutorials/parallel_comp/` (Erişim Tarihi: Şubat 2014).

131. Chen, J., G. Wu, 2002. Parallel Computer Architectures. Higher Education Press.

132. Wilkinson, B. and Allen, M., 1999. Parallel Programming. Upper Saddle River, NJ: Prentice Hall, 460 pp.

133. El-Rewini, H. and Abd-El-Barr, M., 2005. Advanced Computer Architecture and Parallel Processing (Wiley Series on Parallel and Distributed Computing). Wiley-Interscience, 270 pp.

134. Amdahl, G. M., 1967. Validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS Spring Joint Computer Conference*, volume 30, pp. 483–485. Atlantic City, New Jersey.

135. `http://en.wikipedia.org/wiki/Parallel_computing` (Erişim Tarihi: Şubat 2014).

136. Gustafson, J. L., 1988. Reevaluating amdahl's law. **Communications of the ACM**, **31**(5):532–533.

137. Quinn, M. J., 2003. Parallel Programming in C with MPI and OpenMP. McGraw-Hill Education Group, 530 pp.

138. Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain,

R. H., Daniel, D. J., Graham, R. L., and Woodall, T. S., 2004. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *11th European PVM/MPI Users' Group Meeting Proceedings*, pp. 97–104. Budapest, Hungary.

139. Butenhof, D. R., 1997. Programming with POSIX Threads. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 381 pp.

140. Sinha, N. K. and Gupta, M. M., 1999. Soft Computing and Intelligent Systems: Theory and Applications. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 645 pp.

141. Zadeh, L., 1965. Fuzzy sets. **Information and Control**, **8**:338–353.

142. Haykin, S., 1998. Neural Networks: A Comprehensive Foundation. Prentice Hall PTR, Upper Saddle River, NJ, USA, 850 pp.

143. Eiben, A. E. and Smith, J. E., 2003. Introduction to Evolutionary Computing. Springer Verlag.

144. Pham, D. T. and Karaboga, D., 2000. Intelligent Optimisation Techniques, Genetic Algorithms, Tabu Search, Simulated Annealing And Neural Networks. Springer–Verlag, London, 302 pp.

145. Talbi, E. G., 2009. Metaheuristics: From Design to Implementation. Wiley Publishing.

146. `http://tr.wikipedia.org/wiki/Sinir_hucresi` (Erişim Tarihi: Şubat 2014).

147. Zurada, J. M., 1992. Introduction to Artificial Neural Systems. West Publishing Company, New York, 759 pp.

148. Looney, C. G., 1997. Pattern Recognition Using Neural Networks. Oxford University Press, New York, 458 pp.

149. Vas, P., 1999. Artificial Intelligence Based Electrical Machines and Drivers. Oxford University Press, New York, 625 pp.

150. Anderson, D. and McNeil, G., 1992. Artificial neural networks technology. Technical report, Data and Analysis Center for Software.

151. Bianchi, L., Dorigo, M., Gambardella, L. M., and Gutjahr, W. J., 2009. A survey on metaheuristics for stochastic combinatorial optimization. **Natural Computing**, **8**(2):239–287.

152. Olariu, S. and Zomaya, A. Y., 2005. Handbook Of Bioinspired Algorithms And Applications (Chapman & Hall/Crc Computer & Information Science). Chapman & Hall/CRC.

153. Alba, E. and Tomassini, M., 2002. Parallelism and evolutionary algorithms. **IEEE Transactions On Evolutionary Computation**, **6**(5):443–462.

154. Alba, E. and Luque, G., 2005. Parallel Metaheuristics: A New Class of Algorithms. John Wiley Sons, Hoboken, New Jersey, 551 pp.

155. Welten, S., 2008. Parallelization of evolutionary algorithms. Ph.D. thesis, Swiss Federal Institute of Technology Zurich.

156. Cantu-Paz, E., 2001. Migration policies, selection pressure, and parallel evolutionary algorithms. **Journal of Heuristics**, **7**(4):311–334.

157. Skolicki, Z. and De Jong, K. ., 2005. The influence of migration sizes and intervals on island models. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pp. 1295–1302. New York, NY, USA.

158. Tomassini, M., 2005. Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time. Springer Verlag, Secaucus, NJ, USA.

159. Tang, K., Yao, X., Suganthan, P. N., MacNish, C., Chen, Y. P., Chen, C. M., and Yang, Z., 2007. Benchmark functions for the cec 2008 special session and competition on large scale global optimization. Technical report, Nature Inspired Computation and Applications Laboratory, USTC, China, http://nical.ustc.edu.cn/cec08ss.php.

160. Tang, K., Li, X., Suganthan, P. N., and Weise, T., 2009. Benchmark functions for the cec 2010 special session and competition on large scale global optimization. Technical report, Nature Inspired Computation and Applications Laboratory, USTC, China, http://nical.ustc.edu.cn/cec10ss.php.

161. Banharnsakun, A., Achalakul, T., and Sirinaovakul, B., 2010. Artificial bee colony algorithm on distributed environment. In *Second World Congress on Nature and Biologically Inspired Computing*, pp. 13–18. Kitakyushu, Fukuoka, Japan.

162. Hong, Y. S., Ji, Z. Z., and Liu, C. L., 2013. Research of parallel artificial bee colony algorithm based on mpi. **Applied Mechanics and Materials**, **1430**:380–384.

163. Lillieforsa, H. W., 1967. On the kolmogorov-smirnov test for normality with mean and variance unknown. **Journal of the American Statistical Association**, **62**(318):399–402.

164. Gibbons, J. D. and Chakraborti, S., 2011. Nonparametric Statistical Inference. Marcel Dekker, New York, 672 pp.

165. Rao, R., Savsani, V., and Balic, J., 2012. Teaching-learning-based optimization algorithm for unconstrained and constrained real-parameter optimization problems. **Engineering Optimization**, **44**(12):1447–1462.

166. Yildiz, A. R., 2013. Optimization of multi-pass turning operations using hybrid teaching learning-based approach. **The International Journal of Advanced Manufacturing Technology**, **66**:1319–1326.

167. Toğan, V., 2012. Design of planar steel frames using teaching-learning based optimization. **Engineering Structures**, **34**:225–232.

168. Pawar, P. J. and Rao, R. V., 2013. Parameter optimization of machining processes using teaching-learning-based optimization algorithm. **The International Journal of Advanced Manufacturing Technology**, **67**:995–1006.

169. Nayak, M. R., Nayak, C. K., and Rout, P. K., 2012. Application of multi-objective teaching learning based optimization algorithm to optimal power flow problem. **Procedia Technology**, **6**:255–264.

170. Feng, Z., Lei, W., Xinhong, H., Debao, C., and Bin, W., 2013. Multi-objective optimization using teaching-learning-based optimization algorithm. **Engineering Applications of Artificial Intelligence**, **26**(4):1291–1300.

171. Hoseini, M., Hosseinpour, H., and Bastae, B., 2014. A new multi objective optimization approach in distribution systems. **Optimization Letters**, **8**:181–199.

172. Liang, J., Runarsson, T., Mezura-Montes, E., Clerc, M., Suganthan, P., Coello Coello, C., and Deb, K., 2006. Definitions and evaluation criteria for the cec 2006 special sessionon constrained real-parameter optimization. Technical report, School of EEE, Nanyang Technological University, Singapore.

173. Pal, N. R., Bezdek, J. C., and Tsao, E. C. K., 1993. Generalized clustering networks and kohonen's self-organizing scheme. **IEEE Transactions on Neural Networks**, **4**:549–557.

174. Mao, J. and Jain, A. K., 1995. Artificial neural networks for feature extraction and multivariate data projection. **IEEE Transactions on Neural Networks**, **6**(2):296–317.

175. Liao, S. H. and Wen, C. H., 2007. Artificial neural networks classification and clustering of methodologies and applications-literature analysis from 1995 to 2005. **Expert Systems with Applications**, **32**:1–11.

176. Paterlini, S. and Minerva, T., 2003. Evolutionary Approaches for Cluster Analysis, chapter Soft Computing Applications, pp. 167–178. Springer–Verlag, Germany.

177. Younsi, R. and Wang, W., 2004. A new artificial immune system algorithm for

clustering. In Z. Yang, editor, *Intelligent Data Engineering and Automated Learning (IDEAL 2004)*, pp. 58–64. Berlin.

178. Shelokar, P. S., Jayaraman, V. K., and Kulkarni, B. D., 2004. An ant colony approach for clustering. **Analytica Chimica Acta**, **509**:187–195.

179. Omran, M., Engelbrecht, A., and Salman, A., 2005. Particle swarm optimization method for image clustering. **International Journal of Pattern Recognition and Artificial Intelligence**, **19**(3):297–322.

180. Kao, Y. and Cheng, K., 2006. An aco-based clustering algorithm. In M. Dorigo and et al., editors, *5th International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS06)*, pp. 340–347. Berlin.

181. Tsang, C. H. and Kwong, S., 2006. Ant colony clustering and feature extraction for anomaly intrusion detection. **Studies in Computational Intelligence**, **34**:101–123.

182. Paterlini, S. and Krink, T., 2006. Differential evolution and particle swarm optimisation in partitional clustering. **Computational Statistics Data Analysis**, **50**:1220–1247.

183. Niknam, T., Bahmani Firouzi, B., and Nayeripour, M., 2008. An efficient hybrid evolutionary algorithm for cluster analysis. **World Applied Sciences Journal**, **4**(2):300–307.

184. Karaboga, D., Akay, B., and Ozturk, C., 2007. Artificial Bee Colony (ABC) Optimization Algorithm for Training Feed-Forward Neural Networks, chapter Modeling Decisions for Artificial Intelligence, pp. 318–329. Springer Berlin Heidelberg.

185. Karaboga, D. and Akay, B., 2007. Artificial bee colony (ABC) algorithm on training artificial neural networks. In *15th Signal Processing and Communications Applications, (SIU 2007)*, pp. 1–4. Eskisehir, Turkey.

186. Karaboga, D. and Ozturk, C., 2011. A novel clustering approach: Artificial bee colony (ABC) algorithm. **Applied Soft Computing**, **11**:652–657.

187. Shukran, M. A. M., Chung, Y. Y., Yeh, W. C., Wahid, N., and Zaidi, A. M. A., 2011. Artificial bee colony based data mining algorithms for classification tasks. **Modern Applied Science**, **5**(4):217–231.

188. Murphy, P. M. and Aha, D. W., 1994. UCI repository of machine learning databases. In http://www.ics.uci.edu/ mlearn/MLRepository.html. CA: University of California, Department of Information and Computer Science, Irvine.

189. Albayrak, M. and Allahverdi, N., 2011. Development a new mutation operator to solve the traveling salesman problem by aid of genetic algorithms. **Expert Systems with Applications**, **38**(3):1313–1320.

190. Karaboga, D. and Gorkemli, B., 2011. A combinatorial artifical bee colony algorithm for traveling salesman problem. In *Innovations in Intelligent Systems and Applications (INISTA)*, pp. 50–53. Istanbul.

EK-1.

Test Fonksiyonları

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı, Soyadı	:	Rüştü AKAY
Uyruğu	:	T.C.
Doğum Tarihi ve Yeri	:	13 Mayıs 1979, Kayseri
Medeni Durumu	:	Evli
Telefon	:	+90 533 571 08 25
e-posta	:	akay@erciyes.edu.tr
Adres	:	Yenidoğan Mahallesi Toki Kümeevler C3-7 No:26 Talas/KAYSERİ

EĞİTİM

Derece	**Kurum**	**Mezuniyet Tarihi**
Yüksek Lisans	EÜ Müh. Fak. Bilgisayar Müh.	2006
Lisans	EÜ Müh. Fak. Bilgisayar Müh.	2002
Lise	Melikgazi Lisesi, Kayseri	1998

İŞ DENEYİMLERİ

Yıl	**Kurum**	**Görev**
2009- Halen	Erciyes Üniversitesi	Uzman
2002-2009	Şahin Yazılım A.Ş.	Bilgisayar Müh.

YAYINLAR

SCI, SSCI, AHCI Indekslerine Giren Dergilerde Yayımlanan Makaleler

1. Baştürk A., Akay R., "Performance Analysis Of The Coarse-Grained Parallel Model Of The Artificial Bee Colony Algorithm", INFORMATION SCIENCES, vol.253, pp.34-55, 2013

2. Baştürk A., Akay R., "Parallel Implementation Of Synchronous Type Artificial Bee Colony Algorithm For Global Optimization", JOURNAL OF OPTIMIZATION THEORY AND APPLICATIONS, vol.155, no.3, pp.1095-1104, 2012

Hakemli Kongre/Sempozyumların Bildiri Kitaplarında Yer Alan Yayınlar

1. Akay R., Baştürk A., "Gezgin Satıcı Problemi İçin Paralel Yapay Arı Koloni Algoritması", IEEE 22. Sinyal İşleme ve İletişim Uygulamaları Kurultayı, TRABZON, TÜRKIYE, 23-25 Nisan 2014, pp.1-7

2. Akay R., Baştürk A., "Paralel Diferansiyel Gelişim Algoritmasında Farklı Göç Topolojilerinin Performansa Etkisi", Akıllı Sistemlerde Yenilikler ve Uygulamaları Sempozyumu ASYU-2012, TRABZON, TÜRKİYE, 3-4 Temmuz 2012, ss.1-

3. Baştürk A., Akay R., Kalinli A., "Comparison Of Fine-Grained And Coarse-Grained Parallel Models In Particle Swarm Optimization Algorithm", 2nd World Conference on Information Technology WCIT-2011, TÜRKIYE, 23-27 November 2011, pp.1-

4. Baştürk A., Akay R., Kalinli A., Yüksel M.E., "İşaret İşleme Algoritmalarının Gpu Üzerinde Gerçekleştirilerek Hesaplama Sürelerinin Azaltılması", IEEE 19. Sinyal İşleme ve İletişim Uygulamaları Kurultayı - SİU 2011, TÜRKIYE, 20-22 Nisan 2011, pp.262-265