
**T.C.**
**SELÇUK ÜNİVERSİTESİ**
**FEN BİLİMLERİ ENSTİTÜSÜ**

# PARÇACIK SÜRÜ VE KARINCA KOLONİ OPTİMİZASYON ALGORİTMALARININ AÇ GÖZLÜ BİLGİ TAKASI STRATEJİSİ KULLANILARAK PARALELLEŞTİRİLMESİ

**Şaban GÜLCÜ**

**DOKTORA TEZİ**

**Bilgisayar Mühendisliği Anabilim Dalı**

**Ağustos-2017**
**KONYA**

## TEZ KABUL VE ONAYI

Şaban GÜLCÜ tarafından hazırlanan "Parçacık Sürü ve Karınca Koloni Optimizasyon Algoritmalarının Aç Gözlü Bilgi Takası Stratejisi Kullanılarak Paralelleştirilmesi" adlı tez çalışması 24/08/2017 tarihinde aşağıdaki jüri tarafından oy birliği ile Selçuk Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda DOKTORA TEZİ olarak kabul edilmiştir.

| **Jüri Üyeleri** | **İmza** |
|---|---|
| **Başkan**<br>Prof. Dr. Ahmet ARSLAN | ..... |
| **Danışman**<br>Doç. Dr. Halife KODAZ | ..... |
| **Üye**<br>Doç. Dr. Mustafa Servet KIRAN | ..... |
| **Üye**<br>Yrd. Doç. Dr. Ahmet BABALIK | ..... |
| **Üye**<br>Yrd. Doç. Dr. Onur İNAN | ..... |

Yukarıdaki sonucu onaylarım.

Prof. Dr. Mustafa YILMAZ
FBE Müdürü

## TEZ BİLDİRİMİ

Bu tezdeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edildiğini ve tez yazım kurallarına uygun olarak hazırlanan bu çalışmada bana ait olmayan her türlü ifade ve bilginin kaynağına eksiksiz atıf yapıldığını bildiririm.

## DECLARATION PAGE

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Şaban GÜLCÜ

Tarih: 08.08.2017

# ÖZET

## DOKTORA TEZİ

## PARÇACIK SÜRÜ VE KARINCA KOLONİ OPTİMİZASYON ALGORİTMALARININ AÇ GÖZLÜ BİLGİ TAKASI STRATEJİSİ KULLANILARAK PARALELLEŞTİRİLMESİ

**Şaban GÜLCÜ**

**Selçuk Üniversitesi Fen Bilimleri Enstitüsü**
**Bilgisayar Mühendisliği Anabilim Dalı**

**Danışman: Doç. Dr. Halife KODAZ**

**2017, 148 Sayfa**

**Jüri**
**Doç. Dr. Halife KODAZ**
**Prof. Dr. Ahmet ARSLAN**
**Doç. Dr. Mustafa Servet KIRAN**
**Yrd. Doç. Dr. Ahmet BABALIK**
**Yrd. Doç. Dr. Onur İNAN**

Bilim ve teknolojideki hızlı ilerlemeler her alanda olduğu gibi hesaplama alanındaki gelişmeleri de tetiklemektedir. Bu gelişmelerin sonucu olarak, bilgisayarlar tarafından çözülmesi gereken problemler daha büyük ve daha karmaşık olmaktadır ve büyük ölçekli verilerin artmasına neden olmaktadır. Böylece, bu problemleri çözmek ve büyük ölçekli verileri işlemek için paralel hesaplamaya ve paralel algoritmalara ihtiyaç duyulmaktadır.

Optimizasyon problemlerini çözmek için birçok metasezgisel algoritma geliştirilmiştir ve yeni algoritmalar da önerilmektedir. Metasezgisel optimizasyon algoritmaları genellikle seri bir yapıya sahiptir. Metasezgisel optimizasyon algoritmalarının kullanımı problemin çözüm süresini düşürmesine rağmen bazı gerçek dünya problemlerinin çözümleri hala çok uzun zaman almaktadır. Bu yüzden, hem çözüm süresini azaltmak hem de çözümlerin kalitesini artırmak için paralel hesaplamanın metasezgisel algoritmalar ile beraber kullanılması araştırmacıların ilgi duyduğu bir konu olmuştur.

Bu tez çalışmasında Açgözlü Bilgi Takası (ABT) diye adlandırılan yeni bir işbirliği stratejisi geliştirilmiştir. Bu stratejide göç süreci, periyodik olarak belirli bir iterasyon sayısı sonrasında oluşmaktadır. Haberleşmeler sadece sunucu-sürü ile istemci sürüler arasındadır ve istemci sürüler arasında doğrudan bir haberleşme bulunmamaktadır. Bu topolojide düşük haberleşme frekansı olduğu için yüksek hesaplama verimliliği elde edilmektedir. Her bir istemci-sürü kendi lokal en iyi çözümünü sunucu-sürü ile paylaşmaktadır. Daha sonra her bir istemci sürüden rasgele seçilen birer parçacık, sunucu tarafından gönderilmiş olan global en iyi parçacık ile değiştirilmektedir.

Bu tez çalışmasında, sürekli optimizasyon probleminin çözümü için parçacık sürü optimizasyon algoritması (PSO) temelli paralel kapsamlı öğrenme parçacık sürü optimizasyonu (PCLPSO) algoritması ve ayrık optimizasyon probleminin çözümü için karınca koloni optimizasyon algoritması (ACO) temelli paralel karınca koloni optimizasyonu ile birleştirilmiş 3Opt algoritması (PACO-3Opt) önerilmiştir. Bu algoritmalar ABT işbirliği stratejisini kullanmaktadırlar. Önerilen paralel algoritmaları geliştirmek için bir arakatman olan Jade yazılım çatısı kullanılmıştır.

Bu iki yeni paralel metasezgisel algoritmanın sağladığı avantajlar şu şekilde sıralanabilir: çoklu-sürü ve işbirliği özellikleri sayesinde çözüm kalitesini ve global arama kabiliyetini artırmaktadır. Çoklu-sürü tekniğinde, popülasyon alt popülasyonlara bölünmektedir ve her bir alt popülasyon birbirinden bağımsız olarak tüm arama uzayında çalışmaktadır. Alt popülasyonlar en güncel çözümleri takas etmek için ve aramayı daha kaliteli çözümlere doğru yönlendirmek için periyodik olarak işbirliği yapmaktadır.

Algoritmalar dağıtık sistem üzerinde paralel olarak çalışması sayesinde aramayı önemli derecede hızlandırmaktadırlar. Büyük ölçekli problemleri makul süreler içerisinde çözmektedirler.

PCLPSO algoritması 14 tane kıyaslama fonksiyonu üzerinde test edilmiştir ve algoritmanın sonuçları PSO tabanlı 9 farklı algoritmanın sonuçları ile kıyaslanmıştır. Sonuçlar PCLPSO algoritmasının diğer algoritmalardan daha başarılı olduğunu göstermiştir. Ayrıca, PCLPSO algoritması seri versiyonu ile de kıyaslanmıştır. Problem boyutu arttıkça PCLPSO problemleri seri versiyonundan oldukça daha hızlı çözmüştür.

Ayrıca, ABT stratejisinin topolojisi halka, 2B ağ ve tam çizge topolojileri ile karşılaştırılmıştır. Genel olarak deney sonuçlarına bakıldığında, ABT'nin topolojisinin diğer işbirliği stratejilerinin topolojilerinden daha başarılı sonuçlar ürettiği gözlemlenmiştir.

PACO-3Opt algoritması, hem küçük ölçekli hem de büyük ölçekli Gezgin Satıcı Problemleri (GSP) üzerinde test edilmiştir. Sonuçlar PACO-3Opt algoritmasının GSP örneklerinin çözümünde iyi bir performans elde ettiğini göstermiştir. Ayrıca, PACO-3Opt algoritması seri versiyonu ile kıyaslanmıştır. PACO-3Opt algoritması bütün test örneklerini seri versiyonundan oldukça hızlı çözmektedir.

Genel olarak PCLPSO algoritmasının ve PACO-3Opt algoritmasının deneysel sonuçlarını incelediğimizde, bu algoritmalar karşılaştırılan algoritmalardan daha tutarlı ve daha başarılı çözümler elde etmişlerdir.

**Anahtar Kelimeler:** Çoklu-sürü stratejisi, Dağıtık sistem, Gezgin satıcı problemi, Karınca koloni optimizasyonu, Kıyaslama fonksiyonları, Paralel algoritma, Paralel hesaplama, Sunucu-istemci mimarisi.

# ABSTRACT

## Ph.D THESIS

## PARALLELIZATION OF THE PARTICLE SWARM AND ANT COLONY OPTIMIZATION ALGORITHMS BY USING THE GREEDY INFORMATION SWAP STRATEGY

**Şaban GÜLCÜ**

**THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCE OF SELÇUK UNIVERSITY**
**THE DEGREE OF DOCTOR OF PHILOSOPHY**
**IN COMPUTER ENGINEERING**

**Advisor: Assoc. Prof. Dr. Halife KODAZ**

**2017, 148 Pages**

**Jury**
**Assoc. Prof. Dr. Halife KODAZ**
**Prof. Dr. Ahmet ARSLAN**
**Assoc. Prof. Dr. Mustafa Servet KIRAN**
**Asst. Prof. Dr. Ahmet BABALIK**
**Asst. Prof. Dr. Onur İNAN**

Rapid advances in science and technology trigger developments in the computation area as in all areas. As a result of these developments, problems to be solved by computers become larger and more complex and they cause the increments of the big data. Thus, parallel computing and parallel algorithms are needed to solve these problems and to process the big data.

Many metaheuristic algorithms were developed to solve the optimization problems and new algorithms have been also proposed. Metaheuristic optimization algorithms have usually a serial structure. Although their usage reduces time complexity, the solutions of some real-world problems still take too long time. Thus, the usage of the parallel computing together with metaheuristic algorithms both to decrease the search time and to increase the quality of the solutions becomes a popular research area.

In this thesis, a new cooperation strategy called Greedy Information Swap (GIS) was developed. In this strategy, the migration process occurs periodically after a certain number of iterations. The communications are only between the master swarm and the slave swarms and there isn't any directly communication between slave swarms. Because there is a low frequency of communications in this topology, high computational efficiency is obtained. Each slave-swarm shares its own local best solution with the master swarm. Then, a random selected particle in each slave swarm is replaced with the global best solution was sent by the master.

In this thesis, the parallel comprehensive learning particle swarm optimization (PCLPSO) algorithm based on particle swarm optimization (PSO) algorithm was proposed to solve the continuous optimization problem and the parallel ant colony optimization combined with 3Opt algorithm (PACO-3Opt) algorithm based on ant colony optimization (ACO) algorithm was proposed to solve the discrete optimization problem. These parallel algorithms use the GIS cooperative strategy. The Jade software framework which is a middleware is used to develop the proposed parallel algorithms.

The advantages of these two new parallel metaheuristic algorithms can be summarized as follows: they improve the solution quality and the global search ability through their multiswarm and cooperation properties. In the multiswarm technique, a population is divided into subpopulations and each subpopulation explores the whole search area. The subpopulations periodically cooperate to exchange

solutions found up the date and to guide the search toward solutions of better quality. The algorithms significantly decreased the search time through their parallel running on a distributed environment. They solve large-scale problems within reasonable time.

The PCLPSO algorithm has been tested on 14 benchmark functions and the results of the algorithm are compared with the results of 9 different algorithms based on PSO. The results show that the PCLPSO algorithm is more successful than the other algorithms. Besides, the PCLPSO algorithm is also compared with its serial version. As the problem size increases, PCLPSO solves the problems much faster than the serial version.

Furthermore, the topology of the GIS strategy was compared the topologies of ring, 2D mesh and complete graph. In general according to the test results, it has been observed that the topology of the GIS produces more successful results than the topologies of the other cooperative strategies.

PACO-3Opt algorithm was tested with small- and large-scale Travelling Salesman Problems (TSP). The results show that the PACO-3Opt algorithm achieves good performance in solving TSP instances. Besides, the PACO-3Opt algorithm is also compared with its serial version. The PACO-3Opt algorithm solves all the test instances much faster than the serial version.

In general, when we analyze the experimental results of the PCLPSO algorithm and the PACO-3Opt algorithm, these algorithms obtain more consistent and more successful solutions than the algorithms compared.

**Keywords:** Ant colony optimization, Benchmark functions, Distributed system, Master-slave paradigm, Multi-swarm strategy, Travelling salesman problem, Parallel algorithm, Parallel computing.

# ÖNSÖZ

Doktora tez çalışmam boyunca değerli katkılarını, yönlendirici desteğini ve anlayışını hiçbir zaman esirgemeyen danışmanım Sayın Doç. Dr. Halife KODAZ'a, tezin gelişmesine yönlendirici görüş ve önerileri ile yardımcı olan tez izleme komitesi üyeleri Sayın Prof. Dr. Ahmet ARSLAN'a ve Sayın Doç. Dr. Mustafa Servet KIRAN'a, tez çalışmasının son halini alması aşamasında kıymetli fikirleri ile katkıda bulunan jüri üyeleri Sayın Yrd. Doç. Dr. Ahmet BABALIK'a ve Sayın Yrd. Doç. Dr. Onur İNAN'a, bu süre boyunca göstermiş oldukları desteklerinden, ilgilerinden ve yardımlarından dolayı Selçuk Üniversitesi Bilgisayar Mühendisliği Bölümü'nün tüm öğretim elemanlarına ve Necmettin Erbakan Üniversitesindeki çalışma arkadaşlarıma, özellikle manevi desteğini hiç bir zaman esirgemeyen her zaman ve her konuda hep yanımda olan eşime ve aileme teşekkür ederim.

Şaban GÜLCÜ
KONYA-2017

# İÇİNDEKİLER

## SİMGELER VE KISALTMALAR

| | | |
|---|---|---|
| 2Opt | : | 2-opt lokal arama algoritması |
| 3Opt | : | 3-opt lokal arama algoritması |
| ABC | : | Yapay arı koloni algoritması (Artificial Bee Colony) |
| ABT | : | Açgözlü Bilgi Takası |
| ACO | : | Karınca koloni optimizasyon algoritması |
| ACOMAC | : | Çoklu-karınca klan konsepti algoritması |
| AMS | : | Ajan yönetim sistemi (Agent Management System) |
| CC-NUMA | : | Önbellek tutarlı - düzensiz bellek erişimi (Cache Coherent-Non Uniform Memory Access) |
| CC-UMA | : | Önbellek tutarlı - düzenli bellek erişimi (Cache Coherent-Uniform Memory Access) |
| CGAS | : | Kooperatif genetik karınca sistemi |
| CLPSO | : | Kapsamlı öğrenme parçacık sürü optimizasyonu |
| CPSO | : | Kooperatif parçacık sürü optimizasyonu |
| DF | : | Rehber kolaylaştırıcı (Directory Facilitator) |
| DIWO | : | Ayrık istilacı yabani ot optimizasyonunu |
| $d_{ij}$ | : | i ve j şehirleri arasındaki uzaklığı temsil etmektedir |
| DNN | : | İkili en yakın komşu (dual nearest neighbour) |
| EH | : | Elit havuzu |
| FDR-PSO | : | Uygunluk mesafe oranına dayalı parçacık sürüsü optimizasyonu algoritması |
| FE | : | Maksimum uygunluk fonksiyonu değerlendirmesi |
| FIPS | : | Tamamen bilgilendirilmiş parçacık sürüsü algoritmasını |
| GA | : | Genetik algoritma |
| gbest | : | Tüm popülasyondaki global en iyi çözüm |
| gbestTour | : | Tüm popülasyondaki global en iyi tur |
| GSP | : | Gezgin satıcı problemi |
| HACO | : | Çapraz-silme yöntemi ile birleştirilmiş karınca koloni algoritması |
| HCACO | : | Karınca koloni algoritması ve kümeleme algoritmaları tabanlı hiyerarşik algoritma |
| HGA | : | Lokal optimizasyon stratejiye sahip hibrit genetik algoritma |

| | | |
|---|---|---|
| HHNN | : | En yakın komşu algoritması ve en kısa rota ve ekleme algoritmasına dayalı hiper-sezgisel algoritma |
| HMMA | : | Lokal arama algoritmalı hibrit maksimum-minimum karınca sistemi |
| HPSACO | : | Karınca koloni algoritması ve parçacık sürü optimizasyonu tabanlı hibrit sezgisel algoritma |
| IVRS | : | Bireysel varyasyon ve rotalama stratejisi |
| Jade | : | Java ajan geliştirme çatısı (Java Agent DEvelopment Framework) |
| lbest | : | Bir sürüdeki lokal en iyi çözüm |
| lbestTour | : | Bir kolonideki lokal en iyi tur |
| MIMD | : | Çok komut çok veri (multiple instruction multiple data) |
| MISD | : | Çok komut tek veri (multiple instruction single data) |
| MPI | : | Mesaj aktarım arayüzü (Message Passing Interface) |
| MPMD | : | Çok program çok veri (multiple program multiple data) |
| NN | : | En yakın komşu (nearest neighbour) |
| NUMA | : | Düzensiz bellek erişimi (Non-uniform Memory Access) |
| OpenMP | : | Açık kaynak çoklu işleme (Open Multi-Processing) |
| P | : | Göç periyodu |
| PACO-3Opt | : | 3Opt algoritması ile birleştirilmiş paralel karınca koloni optimizasyonu algoritması |
| pbest | : | Parçacığın önceki en iyi çözümü |
| Pc | : | Öğrenme olasılığı |
| PCLPSO | : | Paralel kapsamlı öğrenme parçacık sürü optimizasyonu algoritması |
| PMSOM | : | Kendi-organizeli haritaya dayalı paralelleştirilmiş böl-ve-fethet algoritması |
| PSO | : | Parçacık sürü optimizasyon algoritması |
| PSO-ACO-3Opt | : | Parçacık sürü optimizasyonu, karınca koloni algoritması ve 3-Opt algoritmasına dayalı hibrit bir algoritma |
| RABNET-TSP | : | Sinir ağları ve bağışıklık sistemine dayalı sezgisel algoritma |
| SA | : | Benzetimli tavlama algoritması |
| SIMD | : | Tek komut çok veri (single instruction multiple data) |
| SISD | : | Tek komut tek veri (single instruction single data) |
| SPMD | : | Tek program çok veri (single program multiple data) |
| TSPLIB | : | GSP kütüphanesini temsil etmektedir |

| | | |
|---|---|---|
| UMA | : | Düzenli bellek erişimi (Uniform Memory Access) |
| UPSO | : | Birleştirilmiş parçacık sürü optimizasyonu algoritması |
| *w* sabiti | : | Eylemsizlik ağırlığını temsil etmektedir |
| WFA | : | Su akışı algoritması |

# ŞEKİLLER LİSTESİ

# ÇİZELGELER LİSTESİ

# 1. GİRİŞ

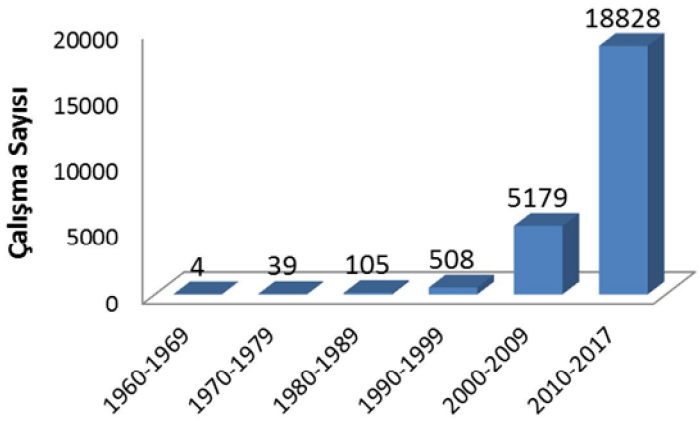
Optimizasyon, mevcut alternatif elemanlar kümesinden bazı kriterlere göre en iyi elemanı seçme olarak tanımlanabilir. Optimizasyon matematik, finans, bilgisayar bilimi ve mühendisliği gibi çeşitli alanlardaki problemlerin çözümü için önemli bir tekniktir. Optimizasyon problemleri genel olarak *sürekli* ve *ayrık* olarak iki gruba ayrılmaktadır (Gould, 2006). Ayrık optimizasyon, değişkenlerin sadece ayrık değer (genellikle tamsayı) alabildiği durumlarla ilgilenir. Bu problemler genellikle çok zordur ve bütün olası çözümlerin teker teker denenmesi ile en iyi çözümün bulunduğu garanti edilir. Kolay problemlerin en iyi çözümü ise basit (açgözlü) sezgisel yöntemlerle de bulunabilmektedir. Ayrık optimizasyonun aksine sürekli optimizasyonda amaç fonksiyonunda kullanılan değişkenlerin, arasında boşluk bulunmayan gerçek değerler kümesinden seçilen sürekli değişkenler olmaları gerekmektedir. Optimizasyon, bir $f(.)$ amaç fonksiyonun minimizasyonu veya maksimizasyonu ile ilgilidir ve minimizasyon problemleri için genel olarak Denklem (1.1) ile ifade edilir:

$$\min_{x \in F} f(x) \tag{1.1}$$

Burada $F$ olası çözümler kümesini temsil etmektedir.

Optimizasyon problemlerinin çözümünde *kesin yöntemler* ve *metasezgisel algoritmalar* kullanılmaktadır. Kesin yöntemler optimum sonucu bulmasına rağmen çok boyutlu, büyük ölçekli gerçek dünya problemlerini çok uzun zamanda çözdüğü için uygulamada çok sık kullanılmazlar. Diğer taraftan, metasezgisel algoritmalar ise optimum sonucu veya optimum sonuca yakın bir sonucu makul bir süre içerisinde buldukları için uygulamada çok sık kullanılırlar. Optimizasyon problemlerini çözmek için birçok metasezgisel algoritma geliştirilmiştir ve yeni algoritmalar da önerilmektedir. Sürü zekâsı ve evrimsel hesaplama, metasezgisel algoritmaların iki popüler alt sınıfıdır. Sürü zekâsı yöntemleri parçacık sürü optimizasyonu (PSO) (Kennedy ve Eberhart, 1995), yapay karınca koloni algoritması (ACO) (Dorigo ve Stützle, 2004), yapay arı koloni algoritması (Karaboga ve Basturk, 2007), boz kurt optimizasyonu (Mirjalili ve ark., 2014) vb. yöntemleri içermektedir. Diğer taraftan, evrimsel hesaplama genetik algoritma (Goldberg, 1989), memetik algoritma (Neri ve ark., 2011) ve gen ifade algoritması (Ferreira, 2006) vb. yöntemleri içermektedir.

Optimizasyon algoritmaları konusunda yapılan çalışmaların yıllara göre dağılımını Şekil 1.1 göstermektedir. Veriler ProQuest (ziyaret tarihi: 18.12.2016) veritabanında "optimization algorithm" anahtar kelimesi ile yapılan aramanın sonucundan elde edilmiştir. Araştırmacıların optimizasyon algoritmaları konusuna olan ilgisinin artmakta olduğunu grafikte net bir şekilde görmekteyiz. 2010-2017 yılları arasında yapılan çalışmaların sayısı 2000-2009 yılları arasında yapılan çalışmaların sayısının üç katından bile fazladır. Diğer bir ifade ile optimizasyon yöntemleri konusu oldukça güncel ve ilgi çekici bir araştırma konusudur.




**Şekil 1.1.** Optimizasyon algoritmaları konusunda yapılan çalışmaların yıllara göre dağılımı

## 1.1. Tezin Amacı ve Literatüre Katkıları

Bilim ve teknolojideki hızlı ilerlemeler her alanda olduğu gibi hesaplama alanındaki gelişmeleri de tetiklemektedir. Bu gelişmelerin sonucu olarak, bilgisayarlar tarafından çözülmesi gereken problemler daha büyük ve daha karmaşık olmaktadır ve büyük ölçekli verilerin artmasına neden olmaktadır. Böylece, bu problemleri çözmek ve büyük ölçekli verileri işlemek için paralel hesaplamaya ve paralel algoritmalara ihtiyaç duyulmaktadır.

Paralel hesaplamada, sonuçları daha kısa sürede elde etmek için alt görevlere ayrılmış bir görev çoklu işlemcilerde eşzamanlı olarak çalıştırılır (Grama, 2003). Paralel hesaplama kullanarak, algoritmanın performansı artmaktadır ve büyük ölçekli problemler daha kısa zamanda çözülmektedir. Bundan dolayı, paralel hesaplama günümüzde tıbbi görüntü işleme (Zhu ve Cochoff, 2010), biyoinformatik (Zomaya, 2006), veri madenciliği (Zaki ve Ho, 2000), finans (Hong ve ark., 2010) gibi birçok alanda kullanılmaktadır. Bu tez çalışmasında, önerilen paralel algoritmaları geliştirmek

için bir arakatman olan Jade yazılım çatısı (Bellifemine ve ark., 2007) kullanılmaktadır. Arakatman; farklı yapıdaki bilgisayarları, ağları ve işletim sistemlerini destekleyen ve ayrıca paralel uygulamalar geliştirmeyi mümkün kılan işletim sistemleri ve uygulamalar arasındaki bir yazılım katmanıdır (Tanenbaum ve Van Steen, 2007).

Metasezgisel optimizasyon algoritmaları genellikle sıralı bir yapıya sahiptir. Onların kullanımı zaman karmaşıklığını düşürmesine rağmen, havacılıkta (Olhofer ve ark., 2001; Hasenjäger ve ark., 2005) ve kimyadaki (Lucasius ve Kateman, 1991) gibi akademide ve endüstride meydana çıkan bazı gerçek dünya problemlerinin çözümleri hala çok uzun zaman almaktadır. Bu yüzden, paralel hesaplama hem çözüm süresini azaltmak hem de çözümlerin kalitesini artırmak için metasezgisel algoritmalar ile beraber kullanılmaktadır (Alba, 2005).

Tez çalışmasında, büyük ölçekli optimizasyon problemleri için çözüm bulma süreci önemli sorulara yol açmaktadır:

(i) Makul bir süre içerisinde problemleri çözmek için ne tür paralel metasezgisel algoritmalar kullanılmalıdır?

(ii) Klasik optimizasyon yöntemleri kullanarak elde edilen sonuçlardan daha iyi sonuçlar paralel metasezgisel algoritma kullanılarak nasıl elde edilebilir?

(iii) Paralel metasezgisel algoritma oluşturmak için ihtiyaç duyulan işbirliği, topoloji, göç seçme ve etkileşim stratejileri nasıl olmalıdır ve bunların parametrelerinin en uygun değerleri nelerdir?

(iv) Bu yöntem lokal optimumlara takılmayı engelleyecek midir?

Bu tez çalışmasının amacı, optimizasyon problemlerini özellikle büyük ölçekli optimizasyon problemlerini çözmek için hem çözüm süresini azaltan hem de çözümlerin kalitesini artıran yeni paralel metasezgisel algoritma geliştirmektir. Tez çalışmasında, sürekli optimizasyon probleminin çözümü için PSO temelli ve ayrık optimizasyon probleminin çözümü için ACO temelli iki yeni paralel metasezgisel algoritma önerilmektedir.

Tez çalışması iki kısımdan oluşmaktadır. Tezin ilk kısmında, sürekli optimizasyon problemlerin bir alt dalı olan tek amaçlı optimizasyon test problemlerinin çözümü için PCLPSO (paralel kapsamlı öğrenme parçacık sürü optimizasyonu) algoritması geliştirilmiştir ve literatüre kazandırılmıştır. Kaynak araştırmasından sonra, parçacık sürü optimizasyon algoritması temelli CLPSO (kapsamlı öğrenme parçacık sürü optimizasyonu) algoritmasının (Liang ve ark., 2006) diğer PSO türevlerinden daha iyi performansa sahip olduğu görülmüştür. PCLPSO algoritması, CLPSO algoritmasının

paralelleştirilmiş şeklidir ve Bölüm 4'de ayrıntılı şekilde anlatılmıştır. PCLPSO algoritmasının sağladığı avantajlar aşağıda sıralanmıştır:

- PCLPSO algoritması çoklu-sürü ve işbirliği özellikleri sayesinde çözüm kalitesini ve global arama kabiliyetini artırmaktadır.
- Çoklu-sürü tekniğinde, popülasyon alt popülasyonlara bölünmektedir ve her bir sürü birbirinden bağımsız olarak tüm arama uzayında çalışmaktadır.
- Sürüler en güncel çözümleri takas etmek için ve aramayı daha kaliteli çözümlere doğru yönlendirmek için periyodik olarak işbirliği yapmaktadır. Ayrıca, Açgözlü Bilgi Takası (ABT) diye adlandırılan yeni bir işbirliği stratejisi literatüre kazandırılmıştır.
- PCLPSO algoritmasının dağıtık sistem üzerinde paralel olarak çalışması sayesinde aramayı önemli derecede hızlandırmaktadır.
- Büyük ölçekli problemleri makul süreler içerisinde çözmektedir.

Sunulan yaklaşım, 14 tane kıyaslama fonksiyonu üzerinde test edilmiştir. Bu fonksiyonlar global optimizasyon üzerine çalışanlar tarafından iyi bilinmektedirler ve genellikle optimizasyon algoritmalarının testi için kullanılmaktadırlar. Sonuçlar literatürdeki PSO tabanlı 9 farklı algoritmanın sonuçları ile kıyaslanmıştır ve sunulan algoritmanın başarılı olduğunu göstermiştir. Ayrıca, PCLPSO algoritması seri versiyonu ile kıyaslanmıştır. Problem boyutu arttıkça PCLPSO problemleri seri versiyonundan oldukça hızlı çözmektedir.

Tezin ikinci kısmında, ayrık optimizasyon problemi olan gezgin satıcı probleminin (GSP) çözümü için PACO-3Opt (paralel karınca koloni optimizasyonu ile birleştirilmiş 3Opt algoritması) algoritması geliştirilmiştir ve literatüre kazandırılmıştır.

GSP tam sonucu veren algoritmalar veya sezgisel yöntemler kullanılarak çözülebilmektedir. Tam sonucu veren algoritmalar büyük ölçekli GSP için uygun değildir. Birkaç sezgisel algoritma büyük ölçekli GSP'nin çözümü için geliştirilmiştir (Dorigo ve Gambardella, 1997; Marinakis ve ark., 2011; Feng ve ark., 2013; Avşar ve Aliabadi, 2015). Sezgisel yaklaşımlar GSP'nin çözümünde iyi sonuçlar elde etmelerine rağmen, lokal optimumlardan başarılı şekilde kaçamamaktadırlar. GSP'nin çözümünde etkili algoritmaların birisi de şüphesiz ki ACO algoritmasıdır. Fakat tek koloniye sahip ACO algoritmasının en büyük eksikliği aramanın erken durgunluğudur (Dorigo ve ark., 1996; Kolli ve Sharvani, 2014). Bunun yanında, GSP'nin çözümünde sezgisel

yaklaşımların çalışma süreleri ne yazık ki çok uzun zaman almaktadır. Bu eksikliklerin üstesinden gelmek ve çözüm kalitesini artırmak için 3-Opt lokal arama algoritmasını kullanan karınca koloni optimizasyonuna dayalı çoklu-koloniye sahip ve paralel çalışan paralel işbirlikçi hibrit bir algoritma (PACO-3Opt) önerilmektedir. Bu algoritmanın temel katkıları şunlardır: Bu yeni yöntem çözüm kalitesini artırma ve işlem süresini azaltma yeteneğine sahiptir. PACO-3Opt algoritması çoklu-koloni ve işbirliği özelliği sayesinde çözümlerin kalitesini ve kararlılığını artırmaktadır. Ayrıca, dağıtık hesaplama ortamında paralel olarak çalışma yeteneğiyle hesaplama süresini düşürmektedir. PACO-3Opt algoritması Bölüm 4'de ayrıntılı şekilde anlatılmıştır.

PACO-3Opt algoritması, hem küçük ölçekli problemler üzerinde hem de büyük ölçekli problemler üzerinde test edilmiştir. Küçük ölçekli problem olarak şehir sayısı 50 ile 200 arasında olan 10 adet simetrik GSP örneği kullanılmıştır ve sonuçlar 14 farklı algoritmanın sonuçları ile kıyaslanmıştır. Büyük ölçekli problem olarak ise şehir sayısı 400 ile 800 arasında olan 11 adet simetrik GSP örneği kullanılmıştır ve sonuçlar 8 farklı algoritmanın sonuçları ile kıyaslanmıştır. Sonuçlar göstermiştir ki PACO-3Opt algoritması küçük-ölçekli ve büyük-ölçekli GSP örneklerinin çözümünde iyi bir performans elde etmiştir. PACO-3Opt algoritması diğer optimizasyon algoritmaları ile oldukça rekabetçidir ve GSP örneklerini çözen herhangi bir meta sezgisel algoritmaya alternatif olabilmektedir. Ayrıca, PACO-3Opt algoritması seri versiyonu ile kıyaslanmıştır. PACO-3Opt algoritması bütün test örneklerini seri versiyonundan oldukça hızlı çözmektedir.

### 1.2. Tezin Organizasyonu

Bu tez çalışması 6 ana bölümden oluşmaktadır ve bu bölümler aşağıdaki şekilde organize edilmiştir.

Birinci bölümde, tez çalışmasının konusu hakkında bir giriş yapılmaktadır. Tez çalışmasının amacına ve bu tez çalışmasının literatüre katkılarına değinilmektedir.

İkinci bölümde, tez konusuyla ilgili literatürdeki mevcut çalışmalar incelenmektedir.

Üçüncü bölümde, tez çalışmasında kullanılan materyal ve yöntemler anlatılmaktadır. Öncelikle paralel hesaplama, dağıtık sistemler ve paralel programlama hakkında detaylı bilgi verilmektedir. Devamında, dağıtık sistem üzerinde paralel programlamayı mümkün kılan ve tezde geliştirilen algoritmaların alt yapısını oluşturan

Jade programlama çatısı anlatılmaktadır. Daha sonra, tez çalışmasında kullanılan optimizasyon algoritmaları anlatılmaktadır. Devamında ise, PCLPSO algoritmasının deneysel çalışmalarında kullanılan kıyaslama fonksiyonları ve PACO-3Opt algoritmasının deneysel çalışmalarında kullanılan gezgin satıcı problemlerin özellikleri tanıtılmaktadır. Son olarak Wilcoxon sıra toplam ve Kruskal-Wallis istatistiksel yöntemleri hakkında ayrıntılı bilgi verilmektedir.

Dördüncü bölümde, tez çalışmasında geliştirilmiş olan paralel metasezgisel algoritmalar ayrıntılı bir şekilde anlatılmaktadır. İlk olarak, sürekli optimizasyon problemlerin çözümü için geliştirilen PCLPSO algoritması anlatılmaktadır. İşbirliği stratejisi kullanan algoritmaların topolojilerinden bahsedilmektedir ve tez çalışmasında geliştirilmiş olan ABT isimli strateji ve özellikleri anlatılmaktadır. Açgözlü Bilgi Takası stratejisinin başarısını etkileyen göç periyodu, topoloji, takas edilecek bilgi ve birleşme poliçesi faktörleri açıklanmaktadır. Daha sonra, ayrık optimizasyon problemlerin çözümü için geliştirilen PACO-3Opt algoritması ayrıntılı bir şekilde açıklanmaktadır.

Beşinci bölümde, geliştirilen algoritmaların deneysel sonuçları sunulmaktadır. Öncelikle PCLPSO algoritmasının kıyaslama fonksiyonların farklı boyutlar üzerindeki deneysel sonuçları verilmektedir. Elde edilen sonuçlar ile literatürdeki algoritmaların sonuçları algoritmaların başarısı ve çalışma süreleri açısından karşılaştırılmaktadır ve PCLPSO algoritmasının daha başarılı olduğu gösterilmektedir. CLPSO algoritmasının paralelleştirilmiş hali olan PCLPSO algoritması CLPSO ile istatistiksel olarak karşılaştırılmaktadır ve bu iki algoritmanın arasında önemli bir fark olduğu gösterilmektedir. Devamında PCLPSO algoritması seri versiyonu ile algoritmaların başarısı ve çalışma süreleri açısından karşılaştırılmaktadır. Ayrıca PCLPSO algoritması seri versiyonu ile istatistiksel olarak karşılaştırılmaktadır ve bu iki algoritmanın arasında önemli bir fark olmadığı gösterilmektedir. Bu bölümün ikinci kısmında ise PACO-3Opt algoritmasının deneysel sonuçları sunulmaktadır. Deneyler problemin boyutuna göre küçük-ölçekli problemler ve büyük-ölçekli problemler olmak üzere iki gruba ayrılmaktadır. Her iki grup için PACO-3Opt algoritmasının deneysel sonuçları ayrı ayrı sunulmaktadır ve literatürdeki algoritmaların sonuçları ile karşılaştırılmaktadır. Ayrıca küçük ölçekli problemlerde, göç periyodunun etkisi süre ve ortalama tur uzunluğu açısından araştırılmaktadır. Devamında PCLPSO algoritması seri versiyonu ile karşılaştırılmaktadır. PACO-3Opt algoritması bütün test örneklerini seri versiyonundan oldukça hızlı çözmektedir.

Son olarak altıncı bölümde, tez çalışmasında elde edilen genel sonuçlar özet olarak verilmektedir. Ayrıca, tez konusuyla ilgilenen araştırmacılara konuyla ilgili ileride yapılabilecek çalışmalar hakkında öneriler verilmektedir.

## 2. KAYNAK ARAŞTIRMASI

Kaynak araştırması bölümünde öncelikle PSO tabanlı popüler algoritmalar incelenmektedir ve devamında sürekli optimizasyon problemlerinin çözümü için geliştirilmiş paralel metasezgisel algoritmalar incelenmektedir. Daha sonra GSP'nin çözümü için geliştirilmiş seri ve paralel algoritmalar hakkında bilgi verilmektedir.

Global optimizasyon problemlerini çözmek için PSO algoritması Kennedy ve Eberhart (1995) tarafından geliştirilmiştir. Ayrıca, daha iyi sonuçlar elde etmek için birçok araştırmacı PSO algoritmasını iyileştirmeye çalışmışlardır ve birçok PSO türevi algoritma önermişlerdir.

PSO algoritmasının saf hali daha çok global arama kabiliyetine sahiptir. Ayrıca, lokal aramanın ve global aramanın her ikisi de farklı problemlerin çözümünde avantaj sağladığı bilinmektedir. Fakat lokal arama ve global arama arasında bir ödün verme vardır. Shi ve Eberhart (1998) eylemsizlik ağırlığı $w$ (inertia weight) diye adlandırılan yeni bir parametreyi algoritmaya eklemişlerdir. Eylemsizlik ağırlığı, lokal ve global arama kabiliyetleri arasındaki denge için önemli bir rol oynamaktadır. PSO'nun performansı eylemsizlik ağırlığı sayesinde daha iyidir. Çünkü eylemsizlik ağırlığı kullanan PSO makul iterasyon sayısı içerisinde global optimumu bulmada daha başarılıdır. Shi ve Eberhart deneylerde sabit ve zamanla değişken olmak üzere farklı eylemsizlik ağırlığı değerleri kullanmışlardır. Eylemsizlik ağırlığı küçük ise ($< 0.8$) PSO daha çok lokal arama algoritması gibidir, büyük ise ($> 1.2$) PSO yeni alanlar keşfetmeyi deneyeceği için daha çok global arama yöntemi eğilimindedir. Eylemsizlik ağırlığı ara bir değerde ($0.8 < w < 1.2$) ise global optimumu bulmada daha çok şansa sahip olacaktır. Deney sonuçlarına göre, eylemsizlik ağırlığı ne kadar büyükse çözüm başlangıç popülasyonuna o kadar az bağımlıdır ve PSO yeni alanlar tarama yeteneğine daha çok sahiptir. Herhangi bir optimizasyon arama algoritması için genellikle, algoritmanın başlangıçta daha fazla keşfe sahip popülasyona sahip olması ve daha sonra lokal alanda derinlemesine arama yapması iyi bir fikirdir. Bu nedenle, Shi ve Eberhart (1998) eylemsizlik ağırlığını sabit olarak kullanmak yerine büyük bir değerden başlatıp (örneğin 1.4) iterasyon boyunca azaltarak sıfıra kadar düşüren bir yöntemi önermektedirler.

Clerc ve Kennedy (2002) daralma katsayısı diye adlandırılan yeni bir parametre tanıtmışlardır. Orijinal PSO'da, parçacıkların hızları parçacıkların yörüngelerini kontrol altında tutabilmek için sınırlandırılmalıdır. Fakat daralma katsayılı PSO hem maksimum

hız ($V_{max}$) sınırına ihtiyaç duymaz hem de daha iyi bir performansa sahiptir. Daralma katsayısı ayrıca, parçacıkların lokal optimumlara yakınsamadaki başarısını artırmaktadır.

Kennedy ve Mendes (2002) farklı topolojilere sahip popülasyonların PSO üzerindeki etkilerini incelemişlerdir. Onlar, haberleşmenin çok hızlı olduğu ve topolojideki düğümlerin aşırı derecede birbirine yakın olduğu bir popülasyonun en iyi sonuca doğru hızlıca hareket edebileceğini belirtmektedirler. Ayrıca bilginin en yavaş iletildiği topolojinin halka topolojisi olduğu ve bu topolojiye sahip popülasyonun global optimumu bulmadaki başarısının diğerlerine göre daha kötü olduğu belirtilmektedirler.

Peram ve ark. (2003) FDR-PSO diye adlandırılan *uygunluk mesafe oranına dayalı parçacık sürüsü optimizasyonu* algoritmasını önermişlerdir. FDR-PSO algoritmasında, her bir parçacık kendisinden daha iyi uygunluk değerine sahip olan komşu parçacıklarının tecrübelerinden öğrenmektedir. Bu yöntem sadece hız güncelleme eşitliğini değiştirmektedir ve eşitliğe en iyi en yakın komşu *nbest* parametresini eklemektedir. Böylece daha geniş bir arama uzayı taranmaktadır ve lokal minimumlara takılma olasılığı düşmektedir.

Mendes ve ark. (2004) FIPS diye adlandırılan *tamamen bilgilendirilmiş parçacık sürüsü* algoritmasını önermişlerdir. Standart PSO'da, bir parçacık sürüdeki en iyi parçacığı (*gbest*) kendisine rehber edinir ve sadece bu parçacıktan etkilenir. FIPS'de ise bütün komşular diğer komşuları etkileyen bir kaynak olabilmektedir. Komşuluklar uygulanan topolojiye göre ayarlanmaktadır. Mendes ve ark. tam çizge, halka, dörtlü küme, piramit ve kare topolojileri üzerinde çalışmışlardır. Tam çizge topolojisinde, bir parçacık sürüdeki bütün parçacıklar ile komşuluğu bulunduğu için bu parçacık diğer bütün parçacıkların bilgilerini kullanmaktadır. Halka topolojisinde ise, bir parçacık sadece kendisinin sağındaki ve solundaki komşu parçacıkların bilgilerini kullanmaktadır.

Van den Bergh ve Engelbrecht (2004) *kooperatif parçacık sürü optimizasyonu* (CPSO) yöntemini önermişlerdir. Bu yöntemde, geniş arama uzayı boyut sayısına göre birkaç küçük arama uzayına ayrılmıştır. Her bir küçük arama uzayı bir-boyutlu problemi optimize eden kendi sürüsüne sahiptir. Böylece, sürü alt sürülere ayrılmıştır. Bu alt sürüler problemin farklı parçalarını işbirliği içerisinde optimize etmektedir ve çözüm vektörü alt sürülerin sonuçlarından oluşmaktadır. Van den Bergh ve Engelbrecht çalışmalarında CPSO-$S_k$ ve CPSO-$H_k$ yöntemlerini de önermişlerdir. CPSO algoritmasında $n$ boyutlu bir problem vektörü $n$ parçaya bölünmekte iken CPSO-$S_k$

algoritmasında ise *n* boyutlu problem vektörü *k* parçaya bölünmektedir. Hibrit bir algoritma olan CPSO-$H_k$ algoritması ise performansı artırmak için CPSO-$S_k$ ve PSO algoritmalarının birlikte kullanım şeklidir.

Parsopoulos ve Vrahatis (2005) UPSO diye adlandırılan *birleştirilmiş PSO* algoritmasını geliştirmişlerdir. Standart PSO algoritmasının komşuluk açısından iki ana versiyonu olduğunu ve bunların lokal versiyon ve global versiyon olduğunu belirtmektedirler. Global versiyonda, tüm sürü her bir parçacığın komşuluğu olarak düşünülmektedir ve lokal versiyonda ise daha küçük komşuluklar kullanılmaktadır. UPSO algoritması lokal ve global versiyonun avantajları olan lokal tarama ve global tarama kabiliyetlerini birleştirerek kullanan bir yöntemdir.

Liang ve ark. (2006) *kapsamlı öğrenme parçacık sürü optimizasyonu* (CLPSO) diye adlandırılan PSO türevi bir algoritma sunmuşlardır. Bir parçacığın hızını güncellemek için, orijinal PSO'daki gibi parçacığın *pbest* bilgisini kullanmak yerine diğer bütün parçacıkların *pbest* bilgilerini kullanan yeni bir öğrenme stratejisi kullanılmaktadır. Bu özellik sürünün çeşitliliğini korumakta ve erken yakınsamayı engellemektedir. Hatta PSO'daki *gbest* pozisyonu CLPSO'da asla kullanılmaz. Bu strateji ile CLPSO daha geniş bir arama uzayını taramaktadır ve global optimumu bulma olasılığı artmaktadır. Bundan dolayı CLPSO, PSO'nun performansını önemli derecede artırmaktadır.

Chen ve ark. (2010) PSO-EO diye adlandırılan PSO ve lokal arama algoritması olan aşırı uç optimizasyonu (EO) algoritması temelli hibrit bir yöntem geliştirmişlerdir. Bu hibrit yöntemde PSO'nun global arama yeteneği ile EO'nun lokal arama yeteneği birleştirilmiştir. EO algoritmasında güçsüz parçacıklar ve bunların en yakın komşuları bir sonraki neslin üretiminde mutasyona uğratılır. PSO-EO yönteminde, algoritmanın başlangıcında kullanıcı tarafından bir *k* sayısı belirlenmektedir ve her *k* iterasyonda bir EO algoritması çalıştırılmaktadır. *k* sayısı 50 civarında bir değerde olursa algoritmanın performansının daha başarılı olduğu gözlenmiştir.

Calazan ve ark. (2014) PSO algoritmasının hızını artırmak için PSO'yu paralelleştirmişlerdir ve FPGA (Alanda Programlanabilir Kapı Dizileri) donanımı üzerinde çalıştırmışlardır. Deneylerde 135 kata kadar hız artışı elde etmişlerdir.

Haklı ve Uğuz (2014) global fonksiyon optimizasyon probleminin çözümü için PSO algoritmasını Levy uçuşu ile birleştirmişlerdir. Levy uçuşu, Levy dağılımını kullanarak rasgele yürüyüş belirleme aşamalarıdır. Levy uçuşu kullanıldığında,

parçacıklar tarafından yapılacak uzun atlamalar sayesinde arama uzayında daha etkili bir arama gerçekleşmektedir.

Tang ve ark. (2015) tüm arama uzayında global optimumu aramak için çok hızlı yakınsama hızına sahip *çoklu-strateji adaptif parçacık sürü optimizasyonu* (MAPSO) algoritmasını geliştirmişlerdir. MAPSO algoritması parçacıkların durumuna göre eylemsizlik ağırlığını dinamik olarak değiştirmektedir. Popülasyonun yüksek orandaki benzerliği bütün parçacıkların *gbest* etrafındaki dar bir alanda hareket etmesi anlamına gelmektedir ve bu nedenle bütün popülasyon lokal optimuma takılmaktadır. Bu durum PSO algoritmasının zayıf noktası olarak gösterilmektedir. Tang ve ark. popülasyonun çeşitliliğini artırmak ve böylece popülasyonu lokal optimumdan kurtarmak için elitist öğrenme stratejisini sunmuşlardır. Elitist öğrenme stratejisi, lokal optimum bölgesindeki birkaç özel parçacığa yardım ederek lokal optimum bölgesinden kurtulmalarını sağlamaktadır. Yani elitist öğrenme stratejisi sadece *gbest* üzerinde değil aynı zamanda *gbest*'e çok yakın özel parçacıklar üzerinde de işlem yapmaktadır.

Jensi ve Jiji (2016), Haklı ve Uğuz (2014) tarafından önerilen yöntemi geliştirmişlerdir. Levy uçuşu öncelikle büyük adımlarla başlamakta ve daha sonra küçük adımlar olarak devam etmektedir.

Son birkaç yılda, *paralel metasezgisel optimizasyonu* konusundaki araştırmaların sayısı artmıştır. Aşağıda bu çalışmaların bazılarına yer verilmektedir.

Fan ve Chang (2009) çok-amaçlı optimizasyon problemlerini çözmek için Pareto baskınlığına ve paralel hesaplama tekniğine dayalı paralel parçacık sürü optimizasyonu algoritması önermişlerdir. Önerilen yöntemde çoklu-sürü stratejisi kullanılmıştır ve sürülerin aralarında haberleşmesi için sunucu-istemci mimarisi uygulanmıştır. Ayrıca, 2010 yılında paralel çoklu-sürü PSO algoritması kullanarak çok büyük ölçekli çok modlu fonksiyonları çözmüşlerdir (Fan ve Chang, 2010). Daha iyi performans elde etmek için arama uzayı eşit ve dinamik olarak parçalanmıştır. Sürüler arasındaki bilgi paylaşımını gerçekleştirmek için mesaj aktarım arayüzü (MPI) kütüphanesi kullanılmıştır. Sunucu sürü global arama için sorumludur istemci sürüler ise lokal arama için sorumludur.

PSO tabanlı bir diğer paralel algoritma Kotinis (2011) tarafından çok-amaçlı optimizasyon problemlerinin çözümü için önerilmiştir. Kotinis PSO'nun performansını artırmak için mutasyon ve elitist stratejisini uygulamıştır. Paralel algoritmada çoklu-sürü kullanılmıştır. Ayrıca, arama uzayı sürü sayısı kadar parçaya bölünmektedir. Her bir sürü arama uzayının farklı parçalarında birbirinden bağımsız olarak rastgele

üretilmektedir. Algoritma sunucu-istemci mimarisi temellidir ve algoritmayı paralelleştirmek için OpenMP kütüphanesi kullanılmıştır.

Li ve Wada (2011) dağıtık sistem üzerinde PSO'nun etkisini artıran ve iletişim gecikmesini etkili bir şekilde düşüren bir paralel PSO algoritması önermişlerdir. Popülasyon işlemci sayısına bölünerek her bir işlemcide çalışacak parçacık sayısı belirlenmektedir. Böylece iş yükü dengelemesi sağlanmaktadır. Haberleşme topolojisi olarak tam çizge kullanılmıştır. Yani her bir sürü kendi *gbest* parçacığını diğer sürüler ile takas etmektedir. Algoritmanın performansını kıyaslama fonksiyonları üzerinde denemişlerdir. Deneylerde 16 bilgisayarlı küme kullanmışlardır. Sürüler arasındaki bilgi paylaşımını gerçekleştirmek için ise MPI kütüphanesini kullanmışlardır.

Chaves-González ve ark. (2011) karmaşık optimizasyon problemlerini çözmek için yedi tane metasezgisel algoritma temelli bir paralel hiper-sezgisel algoritma geliştirmişlerdir. Bu algoritmanın performansını kanıtlamak için algoritmayı telekomünikasyondaki gerçek dünya problemi üzerinde test etmişlerdir. Deneyler 128 çekirdek içeren 16 bilgisayarlı küme üzerinde ve MPI kütüphanesi kullanılarak gerçekleştirilmiştir.

Günümüze kadar birçok çok-amaçlı evrimsel algoritma önerilmiştir ve bunlardan bazıları sonlu elemanlar yöntemleri ve akışkanlar mekaniği gibi kararlı mühendislik tasarım optimizasyonu alanlarında kullanılmaktadır. Kararlı mühendislik tasarım problemlerinin çözümü oldukça zordur ve daha çok hesaplama zamanına ihtiyaç duyulmaktadır. Hesaplama zamanını azaltmak için, Ray ve Smith (2006) kararlı mühendislik tasarım problemini çok-amaçlı optimizasyon problemine dönüştürmüşlerdir ve problemi sunucu-istemci mimarisi kullanarak paralel evrimsel algoritma ile çözmüşlerdir.

Paralel hesaplamada hafıza kullanımına göre iki tip mimari kullanılmaktadır: dağıtık hafıza mimarisi ve paylaşımlı hafıza mimarisi (Tanenbaum ve Van Steen, 2007). Paylaşımlı hafıza mimarisinde, işlemciler sadece bir tane paylaşımlı hafıza ile bağlıdır. Bu mimariye sahip süper bilgisayar diye adlandırılan bilgisayarların maliyeti çok yüksektir. Atiqullah ve Rao (2000) süper bilgisayar üzerinde paralel benzetilmiş tavlama algoritması kullanarak büyük ölçekli kısıtlı global optimizasyon problemlerini çözmüşlerdir. Hesaplama hızı 16 işlemci üzerinde sekiz kata kadar çıkmıştır.

Kampolis ve Giannakoglou (2009) hiyerarşik arama tabanlı dağıtık evrimsel algoritma önermişlerdir. Algoritmayı fonksiyon optimizasyon problemlerinde ve uçak kanadı problemlerinde test etmişlerdir.

Omkar ve ark. (2012) çok-amaçlı tasarım problemlerin çözümü için MPI kullanarak PSO tabanlı bir paralel algoritma geliştirmişlerdir. Algoritmayı IBM p720 kümesi üzerinde çalıştırmışlar ve seri koda göre hesaplama süresini 10 kat düşürmüşler.

Gonsalves ve Egashira (2013) *paralel sürülere dayalı PSO* (PSO-PSO) algoritmasını önermişlerdir. Algoritma çoklu evrimsel aşama ve tekli evrimsel aşamadan oluşmaktadır. Çoklu evrimsel aşamada, popülasyon alt sürülere rasgele bölünmektedir ve alt sürüler arama uzayına dağıtılmaktadır. Her bir alt sürü paralel şekilde bağımsız olarak çalışmaktadır ve her bir sürünün *sbest* (swarm best) parçacığı bulunmaktadır. Belirli iterasyon sonra çoklu evrimsel aşama bitmektedir ve tekli evrimsel aşama başlamaktadır. Alt sürüler *sbest* değerini kendi aralarında paylaşmaktadır ve tüm popülasyondaki *gbest* değerini kaydetmektedir. Tekli evrimsel aşamada, bütün alt sürüler birleşmektedir ve *pbest*, *sbest* ve *gbest* kullanılarak algoritma çalışmaya devam etmektedir. Daha sonra tekrar çoklu evrimsel aşamaya geçilmekte ve aramaya devam edilmektedir. Algoritmanın performansı kıyaslama fonksiyonları üzerinde denenmiştir ve deneyler 256 GB hafızalı 24 çekirdekli bir sunucu üzerinde gerçekleştirilmiştir.

Zhang ve ark. (2015) daha kaliteli çözümler elde etmek için sunucu-istemci mimarisine dayalı çoklu-sürü kullanarak dört PSO türevini birleştiren bir *paralel parçacık sürü optimizasyonu* (PPSO) önermişlerdir. Bu yöntemde, sadece beş işlemci (bir sunucu, dört istemci) kullanılmıştır. Sunucu tüm evrim sürecini kontrol etmek zorundadır ve her bir istemci PSO'nun bir türevini çalıştırmakla sorumludur.

Literatürde gezgin satıcı probleminin çözümü için önerilen birçok çalışma bulunmaktadır. Bunlardan en popüleri ACO (Dorigo ve ark., 1996) algoritmasıdır. Ayrıca, literatürde büyük ölçekli gezgin satıcı problemlerini çözen hibrit ve paralel algoritmalar da bulunmaktadır. Aşağıda bu çalışmaların bazılarına yer verilmiştir.

Tsai ve ark. (2004) GSP'yi çözmek için çoklu-karınca klan konsepti (ACOMAC) algoritmasını tanıtmışlardır. Karınca kolonisi bir kaç klana bölünmektedir. Her bir klan diğerleri ile haberleşmektedir ve lokal minimumlardan kaçmak için bilgileri paylaşmaktadır. Böylece, global optimuma yaklaşabilmektedir. Dahası, Tsai ve ark. büyük ölçekli GSP'leri çözmek için ACOMAC'a çoklu en yakın komşu (NN) ve ikili en yakın komşu (DNN) diye adlandırılan iki yöntemi eklemişlerdir. Birkaç GSP üzerinde ACOMAC'ı karınca koloni sistemi (ACS) (Dorigo ve Gambardella, 1997) ile karşılaştıran deneyler gerçekleştirilmiştir. Deneylerin sonuçlarına göre, ACOMAC

ACS'den daha iyi sonuçlar elde etmiştir. Ayrıca, ACOMAC'a NN veya DNN yönteminin eklenmesi performansı artırmaktadır.

Pasti ve De Castro (2006) RABNET-TSP diye adlandırılan sinir ağları ve bağışıklık sistemine dayalı yeni bir sezgisel algoritma önermişlerdir. Sinir ağı kendi kendine organize olmaktadır ve bir tane katmana sahiptir. RABNET-TSP'nin performansını değerlendirmek için simülasyonlar oluşturulmuştur. RABNET-TSP'nin avantajları; tamamen danışmansız olması ve kullanıcı tarafından ayarlanacak sadece bir tane parametreye sahip olmasıdır.

Masutti ve de Castro (2009) RABNET-TSP algoritmasını modifiye etmişlerdir. Yeni algoritmanın temel özellikleri ön tanımlı dairesel komşuluğa sahip olması, gizli katmanı olmayan ileri beslemeli ağa sahip olmasıdır, danışmansızdır ve rekabetçidir. Ayrıca, büyüme ve budama mekanizmalarını da kullanmaktadır. Diğer yapay sinir yöntemleri ile karşılaştırılmıştır ve birçok deneyde onlardan daha iyi performansa sahip olduğu gözlenmiştir.

Chen ve Chien (2011) benzetimli tavlama algoritmasına, genetik algoritmaya, karınca koloni sistemine ve parçacık sürü optimizasyonuna dayalı yeni bir işbirlikçi yöntem sunmuştur. ACO genetik algoritmanın başlangıç çözümlerini üretmek için kullanılmaktadır. Daha sonra, benzetimli genetik algoritma bu başlangıç çözümlerini geliştirmektedir. Belirli bir iterasyon sonunda, PSO karınca grupları arasında feromon değiştirmek için kullanılmaktadır. TSPLIB'den 25 veri seti deneylerde kullanılmıştır. Algoritma literatürdeki birkaç yöntemle karşılaştırılmıştır. Deneysel sonuçlar bu algoritmanın diğer yöntemlerden daha iyi performansa sahip olduğunu göstermiştir.

Junqiang ve Aijia (2012) çaprazı-silme yöntemine ve ACO'ya dayalı bir hibrit karınca koloni optimizasyonu (HACO) önermişlerdir. Çapraz-silme yöntemi üst üste binen çapraz kenarları engellemek için kullanılmıştır.

Dong ve ark. (2012) *kooperatif genetik karınca sistemi* (CGAS) diye adlandırılan yeni bir hibrit algoritma sunmuşlardır. CGAS GSP'yi işbirlikçi olarak çözmek için genetik algoritma (GA) ve ACO'yu birleştirmiştir. GA ve ACO eşzamanlı ve işbirlikçi olarak çalışmaktadır. GA ve ACO arasındaki bilgi takası her iterasyonun sonunda gerçekleşmektedir. Bu takas bir sonraki iterasyon için en iyi sonuçların seçimini sağlamaktadır. Bu işbirliği sayesinde, algoritma global optimuma ulaşmak için daha fazla şansa sahip olmaktadır.

Jun-man ve Yi (2012) gelişmiş bir ACO olan bireysel varyasyon ve rotalama stratejisini (IVRS) sunmuştur. Rotalama stratejisinde, en kısa yolu ilk önce bulan

karınca bütün şehirleri gezmektedir, daha sonra diğer karıncalar bütün şehirleri gezmektedir. Bireysel varyasyon stratejisinde ise bütün karıncalar farklı bir rotalama stratejisine sahiptir. Daha çok çalışan karıncanın feromon etkisi daha fazla olmaktadır. IVRS yöntemi çözümlerin kalitesini artırmak için her bir döngünün sonunda 2-Opt lokal arama algoritmasını uygulamaktadır.

Peker ve ark. (2013) ACO ve taguçi yöntemine dayalı yeni bir algoritma önermişlerdir. Taguçi yöntemi ACO'nun parametrelerini optimize etmek için kullanılmaktadır.

Othman ve ark. (2013) 2-Opt ve 3-Opt lokal arama algoritmaları ile birleştirilmiş *su akışı algoritmasına* (WFA) dayalı yeni bir yöntem sunmuştur. Bu yöntemde popülasyon boyutu dinamik olarak değişmektedir. 2-Opt ve 3-Opt algoritmaları lokal optimumlardan kurtulmak için kullanılmaktadır.

Gündüz ve ark. (2014) ACO algoritması ve yapay arı koloni (ABC) algoritması temelli yeni bir hiyerarşik yöntem (ACO-ABC) önermişlerdir. ACO güzergâh yapımı için sorumludur ve ABC güzergâh iyileştirme için sorumludur. Böylece, en uygun veya en uyguna yakın çözümler elde edilmektedir. Deneylerde 10 adet GSP örneği kullanılmıştır ve önerilen algoritma ACO ve ABC'nin tekil yöntemlerinden daha iyi performansa sahiptir.

Jiang ve ark. (2014) büyük-ölçekli GSP'yi çözmek için ACO ve kümeleme algoritmaları tabanlı hiyerarşik bir algoritma (HCACO) önermişlerdir. Öncelikle, kümeleme algoritmaları büyük-ölçekli problemi küçük-ölçekli şehir kümelerine (yaklaşık 40 şehirli) bölmek için tekrar tekrar uygulanmaktadır. Daha sonra, bütün alt problemler ACO tarafından çözülmektedir. Son olarak, her bir alt problemin çözümleri büyük-ölçekli problemin çözümü olarak birleştirilmektedir. Kullanılan kümeleme algoritmaları *K-ortalamalar* ve *benzerlik yayılması* algoritmalarıdır. Büyük veriler üzerindeki hesaplama maliyetinin düşük olmasından dolayı 3000 şehirden büyük problemler için K-ortalamalar algoritması kullanılmaktadır. 3000 şehirden küçük problemler için ise benzerlik yayılması algoritması kullanılmaktadır.

Wang (2014) iki lokal optimizasyon stratejisine sahip hibrit bir genetik algoritma (HGA) sunmuştur. Birinci lokal optimizasyon stratejisi dört şehirli ve üç hatlı lokal eşitsizliktir ve kısa rotaya sahip lokal yollar oluşturmaktadır. Daha sonra ikinci lokal optimizasyon stratejisi ikiden fazla şehre sahip yollara uygulanmaktadır. Elde edilen lokal yollar genetik algoritma ile birleştirilmektedir.

Yong (2015) lokal arama algoritmalı hibrit bir maksimum-minimum karınca sistemi (HMMA) önermiştir. Öncelikle, maksimum-minimum karınca sistemi optimale yakın bir çözüm bulmak için uygulanmaktadır. Daha sonra, bulunan çözümdeki dört komşu düğümün lokal rotası daha iyi bir tahmin elde etmek için dört düğümlü ve üç hatlı lokal optimal rotaya dönüştürülür. HMMA küçük- ve büyük-ölçekli GSP örnekleri ile test edilmiştir.

Kızılateş ve ark. (2015) *en yakın komşu algoritması* ve e*n kısa rota ve ekleme algoritmasına* dayalı yeni bir hiper-sezgisel algoritma (HHNN) önermişlerdir. Öncelikle, en yakın komşu algoritması ile birkaç şehirli bir tur oluşturulmaktadır. Daha sonra, en kısa rota ve ekleme algoritması ile seçilmemiş şehirlerin tura eklenmesiyle tur genişletilmektedir.

Avşar ve Aliabadi (2015) kendi-organizeli haritaya dayalı paralelleştirilmiş böl-ve-fethet yöntemini (PMSOM) önermişlerdir. Öncelikle, şehirler belediyelere bölünmektedir. Daha sonra, her bir belediye tarafından en uygun çözüm değerlendirilmektedir. Son olarak, komşu belediyeler nihai çözümü oluşturmak için birleştirilmektedir. Önerilen yöntem büyük ölçekli problemlerin çözümünde kullanılmıştır.

Zhou ve ark. (2015) standart *istilacı yabani ot optimizasyonu* algoritmasının gelişmiş versiyonu olan *ayrık istilacı yabani ot optimizasyonunu* (DIWO) önermişlerdir. İlk olarak, ot bireyleri rotalar oluşturmaktadır ve daha sonra uygunluk değerlerini hesaplamaktadırlar. İkinci olarak, 3-Opt lokal arama algoritması kullanılmaktadır. Son olarak, gelişmiş 2-Opt (I2Opt) GSP'yi çözmek için ikinci lokal arama operatörü olarak kullanılmaktadır.

Jia (2015) ACO ve PSO tabanlı yeni bir sezgisel algoritma (HPSACO) sunmuştur. PSO başlangıç çözümlerini elde etmek için kullanılmaktadır. Daha sonra, ACO optimal sonucu elde etmek için uygulanmaktadır.

Mahi ve ark. (2015) PSO-ACO-3Opt diye adlandırılan ACO, PSO ve 3-Opt algoritmalarına dayalı hibrit bir yöntem önermişlerdir. PSO-ACO-3Opt algoritmasında, PSO ACO'nun parametrelerini optimize etmektedir ve 3-Opt lokal optimumlara takılan çözümlerin kalitesini artırmaktadır. Literatürde iyi bilinen birkaç algoritma ile karşılaştırılmaktadır ve onun performansının birçok durumda diğerlerinden daha iyi olduğu görülmektedir.

Pedemonte ve ark. (2011) paralel karınca koloni algoritmaları üzerine kapsamlı bir inceleme yapmışlardır. Günümüzde, grafik işleme birimlerinin (GPU) kullanımı

paralel hesaplamada çok popüler olmuştur. Yakın geçmişte, GPU üzerinde çalışan paralel karınca koloni optimizasyonu önerilmiştir (Delévacq ve ark., 2013). Bu algoritma gezgin satıcı problemine uygulanmıştır ve algoritmanın seri versiyonundan 23 kat daha hızlı çalışmıştır.

# 3. MATERYAL VE METOT

Bu bölümde öncelikle paralel hesaplama, dağıtık sistemler ve paralel programlama hakkında detaylı bilgi verilmektedir. Devamında, dağıtık sistem üzerinde paralel programlamayı mümkün kılan ve tezde geliştirilen algoritmaların alt yapısını oluşturan Jade programlama çatısı anlatılmaktadır. Daha sonra, tez çalışmasında kullanılan optimizasyon algoritmaları anlatılmaktadır. Devamında ise, PCLPSO algoritmasının deneysel çalışmalarında kullanılan kıyaslama fonksiyonları ve PACO-3Opt algoritmasının deneysel çalışmalarında kullanılan gezgin satıcı problemlerin özellikleri tanıtılmaktadır. Son olarak Wilcoxon sıra toplam ve Kruskal-Wallis istatistiksel yöntemleri hakkında ayrıntılı bilgi verilmektedir.

## 3.1. Paralel Hesaplama

Bilim ve teknoloji alanındaki ilerlemeler her alanda olduğu gibi hesaplama alanındaki gelişmeleri de tetiklemektedir. Bu gelişmeler sonuncunda bilgisayarlar tarafından çözülmesi gereken problemler daha büyük ve daha karmaşık hale gelmektedir. Bu problemlerin çözümü için daha hızlı bilgisayarlara ihtiyaç duyulmaktadır. Bu ihtiyacı karşılamak için süper bilgisayarlar ve dağıtık sistemler oluşturulmuştur. Bu bölümde dağıtık sistemler, paralel hesaplama ve paralel programlama hakkında bilgiler verilmiştir.

### 3.1.1. Dağıtık sistemler

Dağıtık sistemler; belirli bir hedefe ulaşmak, belirli bir problemi çözmek için bir ağ üzerinde birbirleriyle iletişim halindeki bilgisayarlardır. Dağıtık hesaplama, bilgisayar biliminde dağıtık sistemleri inceleyen bir bilim dalıdır. Dağıtık hesaplamada dağıtık sistemler kullanılarak problemlerin çözülmesi gerçekleştirilir. Dağıtık hesaplamalarda bir problem birden fazla parçaya bölünür ve problemin her bir parçası farklı bir bilgisayar tarafından çözülür. Dağıtık sistemler, sistemin özelliklerine ve kullanım alanlarına göre birkaç alt başlığa ayrılmaktadır. Bunlardan bir kaçı yüksek başarımlı hesaplama, grid hesaplama, bulut hesaplama şeklinde sıralanabilir.

### 3.1.2. Paralel hesaplama ve uygulama alanları

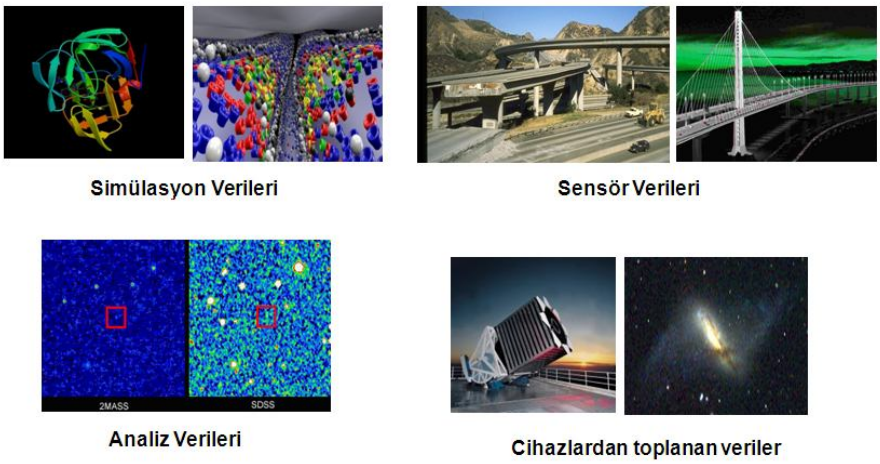
Son 30 yılda mikroişlemci teknolojisinde muazzam ilerlemeler olmuştur. İşlemcilerin saat hızları yaklaşık 40 MHz'den (örneğin 1988'deki MIPS R3000) 5.5 GHz'e (IBM zEC12) kadar artmıştır. Bununla beraber, günümüzde işlemciler aynı döngü içerisinde birden fazla komutu çalıştırma yeteneğine sahiptirler. Aynı dönemde bazı diğer konular da önemli hale gelmiştir. Belki de bunların en belirgin olanı, ihtiyaç duyulan oranda işlemciye veri sağlayan bellek sisteminin yeteneğidir. Ayrıca, mimarideki ve yazılımdaki önemli yenilikler de veri yolu ve bellek tarafından oluşturulan darboğazların hafifletilmesini sağlamıştır (Grama, 2003).

Hesaplama elemanlarını hızlandırma konusundaki eşzamanlılığın rolü uzun yıllardır kabul görmektedir. Özellikle, eşzamanlılığın ölçeklenebilir performans, zaman tasarrufu ve düşük maliyet konularındaki başarısı paralel hesaplama uygulamalarının geniş yelpazesinde görülmektedir (Grama, 2003). Bunun sonucunda, birbirine bağlı iki, dört, hatta sekiz işlemcili masaüstü bilgisayarları ve sunucular yoğun işlem gerektiren uygulamalar için ortak platformlar haline gelmektedir. Bilim ve mühendislikteki büyük ölçekli uygulamalar, genellikle yüzlerce işlemciden oluşan paralel bilgisayarlara bel bağlamaktadır. Veri tabanı veya web sunucuları gibi veri yoğunluğu olan platformlar ve veri madenciliği uygulamaları genellikle yüksek bant genişliği sağlayan iş istasyonları kümesini kullanmaktadır. Grafik ve animasyon alanındaki uygulamalar (Bui ve ark., 2013), milyonlarca poligonlu gerçekçi ortamları hesaplamak ve işlemek için gerçek zamanlı olarak birden fazla işlem elemanı kullanmaktadır. Yüksek düzeyde erişilebilirliğe ihtiyaç duyan programlar da yedekleme için paralel ve dağıtık platformlara bel bağlamaktadırlar. Bu nedenle maliyet, performans ve uygulama gereksinimleri açısından, paralel platformların ilkelerini, araçlarını ve tekniklerini anlamak son derece önemlidir.

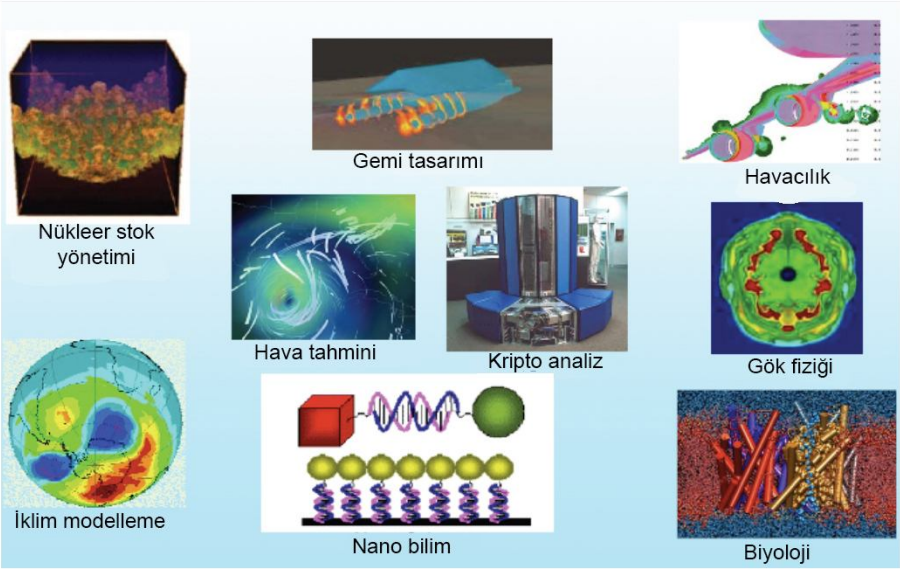
Paralel hesaplama günümüzde birçok alanda kullanılmaktadır. Yaşadığımız bilgi çağında, işlenmesi gereken veriler gün geçtikçe artmaktadır. Şekil 3.1'de gösterildiği gibi veriler sensörlerden, bilimsel çalışmalardan/deneylerden, simülasyonlardan, analizlerden olmak üzere çok çeşitli yerlerden toplanmaktadır. Bu toplanan büyük ölçekli veriler bilim adamları, akademisyenler, analistler olmak üzere birçok kişi tarafından kullanılmaktadır. Ayrıca bilim ve teknoloji ilerledikçe araştırılması gereken konular daha da detaylanmakta ve daha da artmaktadır. Bunun sonucunda da tıp, finans ve astronomi gibi bazı bilim dallarında paralel hesaplamaya olan ihtiyaç gün geçtikçe

artmaktadır. Şekil 3.2'de gösterildiği gibi paralel hesaplamanın birkaç uygulama alanı şunlardır:

- Tıbbi görüntü işleme: tomografi, 3B ultrason gerçek zamanlı X-ray (Kadah ve ark., 2011),
- Finans: black scholes modeli, Monte Carlo simülasyonları (Lee ve ark., 2010),
- Petrol ve Gaz: rezerve modelleme, sismik veri yorumlama (Zhang ve ark., 2001),
- Biyobilim: gen ve protein açıklama, bireyin genlerine uygun ilaç tedavisi, nanobilim, biyoloji (Zomaya, 2006),
- Veri ambarları, veri madenciliği (Tsai ve ark., 2016),
- Akışkanlar mekaniği (Kim ve ark., 2015), gemi tasarımı, hava tahmini, iklim modelleme (Chen ve ark., 2017),
- Nükleer stok yönetimi, askeriye, veri sıkıştırma, arama, güvenlik, kriptoloji, vb.




**Şekil 3.1.** Paralel hesaplamada işlenen veriler (Llnl, 2016)




**Şekil 3.2.** Paralel hesaplamanın birkaç uygulama alanı (Llnl, 2016)

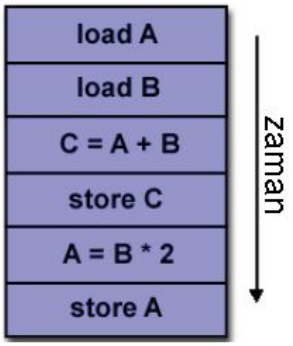
### 3.1.3. Paralel hesaplama modelleri

Paralel hesaplama sistemlerinin sınıflandırılmasında pek çok farklı metot izlenmiştir. En yaygın kullanılan sınıflandırmalardan bir tanesi Flynn (1972) tarafından önerilen Flynn sınıflandırmasıdır. Bu sınıflandırma çok işlemcili bilgisayar mimarilerini komut ve veri olmak üzere iki boyutta incelemektedir. Bu boyutlardan her biri, tekli veya çoklu olası durumlardan sadece birine sahip olabilir. Buna göre olası 4 grup oluşmaktadır:

- Tek komut tek veri (single instruction single data, SISD)
- Tek komut çok veri (single instruction multiple data, SIMD)
- Çok komut tek veri (multiple instruction single data, MISD)
- Çok komut çok veri (multiple instruction multiple data, MIMD)

Bu modellerin dışında tek program çok veri (SPMD) ve çok program çok veri (MPMD) hesaplama modelleri de mevcuttur.

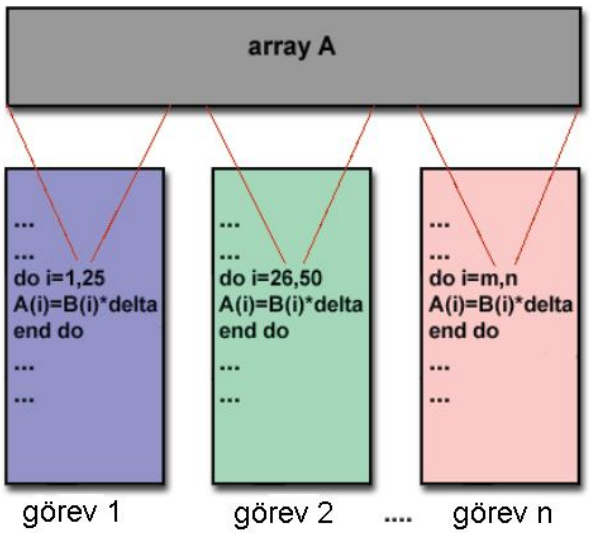
Tek komut tek veri (SISD) seri bilgisayar olarak da düşünülebilir. Şekil 3.3'de gösterildiği gibi merkezi işlem birimi (CPU) üzerinde, her bir cpu saat döngüsünde sadece bir komut çalıştırılabilir ve yine aynı şekilde, her bir cpu saat döngüsünde işlenen veri sayısı da en fazla bir olabilir. Bilgisayarlarda kullanılan en eski fakat en yaygın sistemdir.




**Şekil 3.3.** Tek komut tek veri modelin çalışması (Llnl, 2016)

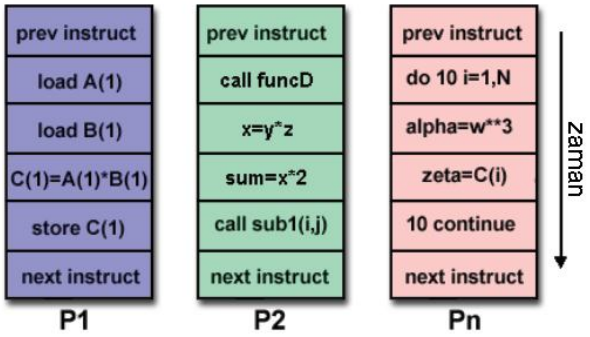
Tek komut çok veri (SIMD) modelinde, Şekil 3.4'de gösterildiği gibi, çalıştırılan her bir paralel işlem (process) aynı cpu saat döngüsünde aynı komutu çalıştırmaktadır. Fakat komutun verileri farklı farklıdır. Yoğunluklu olarak paralel genişliği yüksek haberleşme ağlarında ve düşük kapasiteli komut kümelerine sahip büyük dizilerde kullanılır. Örneğin görüntü işlemede sıklıkla kullanılmaktadır.

Çok komut tek veri (MISD) modelinde, tek bir veri grubunun birden fazla işlem tarafından kullanılmasıyla meydana gelmektedir.




**Şekil 3.4.** Tek komut çok veri modelin çalışması (Llnl, 2016)

Çok komut çok veri (MIMD) modeli paralel hesaplamanın en yaygın türüdür. Modern bilgisayarlar bu türe girmektedir. Şekil 3.5'de gösterildiği gibi her bir işlemci farklı bir komut setini çalıştırabilmektedir ve farklı bir veri kümesiyle çalışabilmektedir. Süper bilgisayarlar, dağıtık sistemler ve bazı kişisel bilgisayarlar bu türe girmektedir.



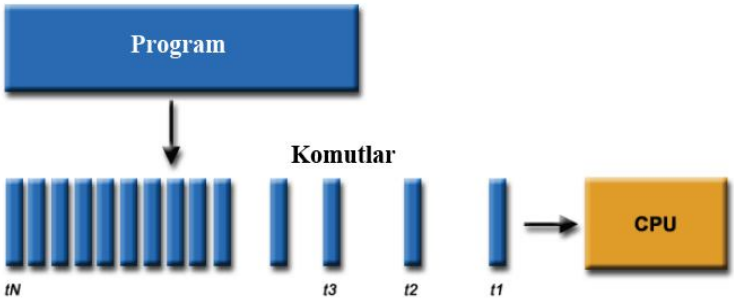

**Şekil 3.5.** Çok komut çok veri modelin çalışması (Llnl, 2016)

Tek program çok veri modeli (SPMD) yüksek seviyeli bir paralel programlama modelidir. SIMD ve MIMD metotlarının bir kombinasyonudur. Tek bir programın eş zamanlı olarak birçok örneği çalıştırılır. Bu programların her biri herhangi bir anda herhangi bir komutun duruma göre tamamını veya belirli bir kısmını yürütebilir. Her program aynı veya farklı veriler üzerinde işlem yapabilir.

Çok program çok veri modeli (MPMD) SPMD modeli gibi yüksek seviyeli bir paralel programlama modelidir. Smpd modelinden farklı olarak MPMD modelinde birbirinden farklı birçok işlem (process) mevcuttur. Farklı işlemler eş zamanlı çalışırken smpd modelinde olduğu gibi aynı işlemin de birden fazla örneği çalışıyor olabilir ve her işlem farklı veri kullanabilir.

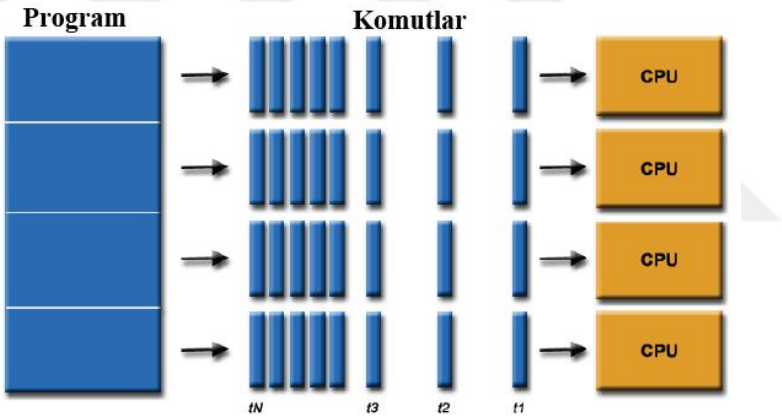
### 3.1.4. Paralel programlama

Bilgisayar programları genellikle seri hesaplama yapmak üzere tek bir bilgisayarda çalıştırılmak için yazılırlar. Şekil 3.6'da gösterildiği gibi seri kodda komutlar sıralı olarak çalıştırılırlar. İşlem sırası gelen komut CPU'ya gönderilir. CPU bu komutu çalıştırır. Bu komut çalıştırılırken sıradaki komutlar bu komutun çalıştırılıp bitmesini beklemek zorundadır.




**Şekil 3.6.** Seri hesaplama (Llnl, 2016)

Paralel hesaplamada durum seri hesaplamadan biraz farklıdır. En basit anlamda paralel hesaplama, bir problemi çözmek için birkaç bilgisayar kaynağının eş zamanlı olarak kullanımıdır. Şekil 3.7'de görüldüğü gibi program öncelikle eş zamanlı olarak çözülecek komut gruplarına ayrılır. Her komut grubu seri komutlardan oluşmaktadır. Her bir parçanın komutları, farklı CPU'lar üzerinde eş zamanlı olarak çalıştırılır.



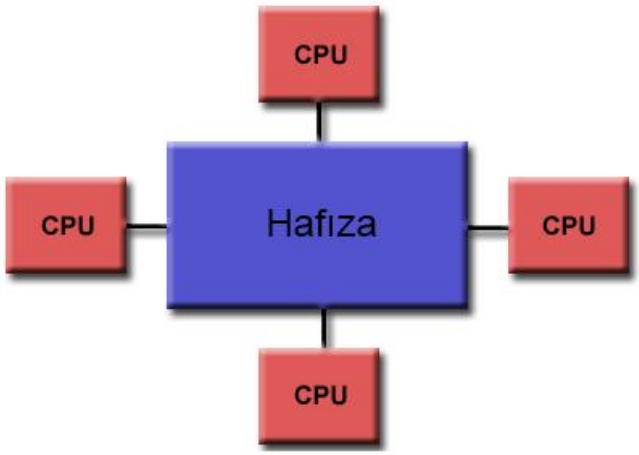

**Şekil 3.7.** Paralel hesaplama (Llnl, 2016)

Paralel hesaplamada birden fazla işlemciye sahip bir bilgisayar kullanılacağı gibi aynı ağa bağlı birden fazla bilgisayar da kullanılabilir. Daha yüksek işlem hacmine ulaşmak için bu sistemlerin birleştirilmesi ile hibrit sistemler de oluşturulabilmektedir.

### 3.1.4.1. Paralel bilgisayar hafıza modelleri

Paralel bilgisayarlar hafızayı kullanma yöntemlerine göre iki gruba ayrılırlar. Bunlar paylaşımlı hafıza ve dağıtık hafıza modelleridir. Bu iki modelin beraber kullanılması ile hibrit bir sistem de oluşturulabilir. Böylece performans artışı elde edilmiş olunur.

**Paylaşımlı hafıza (shared memory):** Paylaşımlı hafıza; aralarında iletişimi sağlamak veya gereksiz kopyaları önlemek için aynı anda birden çok program tarafından erişilebilen hafızadır (Foster, 1995). Paylaşımlı hafıza, programlar arasındaki veri aktarımını etkin bir şekilde gerçekleştirir. Programlar bir tek işlemcide veya birden çok ayrı işlemci üzerinde çalıştırılabilir. Şekil 3.8'de de görüldüğü gibi bu tür hafıza kullanan işlemcilerden her biri hafızanın tamamını kullanabilmektedir.




**Şekil 3.8.** Paylaşımlı hafıza modeli (Llnl, 2016)

Bu modelde işlemciler bağımsız olarak işlemlerini yürütürken aynı hafızayı paylaşmaktadırlar. Böylece bir işlemcinin değişiklik yaptığı hafıza bölgesi diğer işlemciler tarafından da görülebilmekte ve değiştirilebilmektedir. Hafızada paylaşımlı adres alanı kullanmak programcılık açısından büyük kolaylıklar sağlamaktadır. Ayrıca hafızanın CPU ile yakınlığı; hız artışı ve kolay veri paylaşımı sağlamaktadır.

Bu modelde çok yüksek kapasiteli hafızalar kullanılmaktadır. Fakat bazı problemlerde bu hafızalar yeterli kalmayabilir. Bu durumda dağıtık hafıza modeli kullanılmalıdır.

Paylaşımlı hafızaya sahip bilgisayar mimarileri, hafızaya erişim zamanlarına bağlı olarak iki sınıfa ayrılır: Düzenli bellek erişimi (Uniform Memory Access, UMA) ve düzensiz bellek erişimi (Non-uniform Memory Access, NUMA)

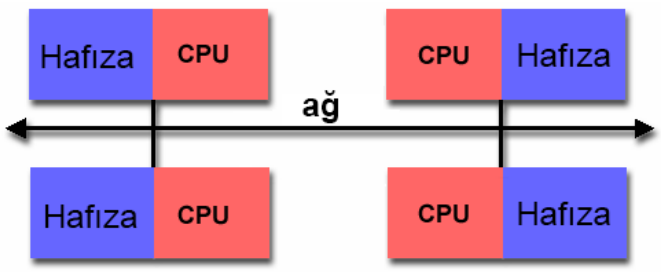
UMA mimarisinde bütün işlemciler hafızaya aynı hızda erişirler. Aynı tip işlemciler kullanılır. Bazen önbellek tutarlı UMA (Cache Coherent UMA, CC-UMA) olarak adlandırılır. Önbellek tutarlılığı (Cache Coherent) donanım seviyesinde olmaktadır ve şu şekilde açıklanabilir: İşlemciler ana bellekteki komutları daha hızlı çalıştırmak için ana bellekteki bir hafıza bloğunu kendi önbelleğine kopyalamaktadır.

Eğer bir işlemci kendi önbelleğindeki bir değişkenin değerini değiştirirse diğer işlemciler bu değişiklikten haberdar olmalıdır ve güncel verinin işlenebilmesi için önbelleklerindeki aynı değişkenin değerinin de değiştirilmesi gerekmektedir. Önbellek uyumluluk yöntemi sayesinde bütün işlemciler bu değişiklikten haberdar olmaktadır.

NUMA mimarisinde ise bütün işlemciler hafızaya farklı hızda erişirler. Önbellek tutarlılığı bu hafıza mimarisinde de vardır ve CC-NUMA olarak adlandırılır.

Paylaşımlı hafıza mimarisinin önemli avantajları şunlardır: Paralel program geliştirme açısından büyük kolaylıklar sağlamaktadır. Hafızanın CPU'ya yakın olmasından dolayı hız artışı sağlamaktadır. Dezavantajları ise şunlardır: Bazı problemlerde hafıza yeterli gelmeyebilir. Paylaşımlı hafızaya sahip daha fazla işlemcili bilgisayarları tasarlamak ve üretmek giderek daha zor ve pahalı hale gelmektedir.

**Dağıtık hafıza (distributed memory):** Dağıtık hafıza mimarisinde, verilerin işlemciler arasında paylaşımı için bir haberleşme ağına ihtiyaç vardır (Şekil 3.9). Bu mimaride her işlemcinin kendi yerel hafızası vardır. Bu hafıza, paylaşımlı değildir ve diğer işlemciler tarafından ulaşılamaz. Her işlemci kendi yerel hafızasına sahip olduğu için diğer işlemcilerden bağımsız olarak işlem yapabilmektedir. Bu nedenle önbellek tutarlılığı sistemine gerek yoktur. İşlemcilerin birisi bir başka işlemcinin verisine ulaşmak isterse, bu haberleşmenin ne zaman ve ne şekilde olacağına programcı karar vermelidir. Veri transferi için makineler arasında Ethernet haberleşme teknolojisi kullanıldığı gibi genellikle çok hızlı iletişimi sağlayan InfiniBand haberleşme teknolojisi kullanılmaktadır.

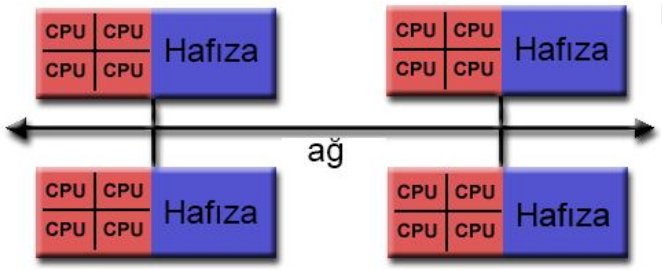



**Şekil 3.9.** Dağıtık hafıza modeli (Llnl, 2016)

Bu mimarinin avantajları şunlardır: Hafıza, işlemci sayısı ile ölçeklenebilir. İşlemci sayısının artması ile hafızanın boyutu da orantılı şekilde artmaktadır. Her bir işlemci önbellek tutarlılığı kontrolüne gerek duymadan kendi hafızasına hızlıca ulaşabilmektedir. Standart işlemciler ve mevcut ağ yapısı kullanılabildiğinden dolayı

maliyeti düşüktür. Ayrıca problemin büyüklüğüne göre kullanılan işlemci adedi yüzleri, binleri bulmaktadır. Dezavantajları ise; dağıtık hafıza sistemlerinde program geliştirmek oldukça zahmetli bir iştir ve hafızalar arası veri paylaşımı çok zaman almaktadır.

**Dağıtık paylaşımlı hafıza (distributed shared memory):** Dağıtık paylaşımlı hafıza mimarisinde, paylaşımlı hafıza ve dağıtık hafıza mimarileri beraber kullanılmaktadır (Şekil 3.10). Günümüzde en hızlı ve gelişmiş paralel hesaplama sistemleri bu mimariyi kullanmaktadırlar. Bu mimaride, her bir düğüm paylaşımlı hafıza modeline sahip çok işlemcili bilgisayardan meydana gelirken, bu bilgisayarlar bir haberleşme ağı ile birleştirilerek dağıtık hafıza sistemini meydana getirirler.



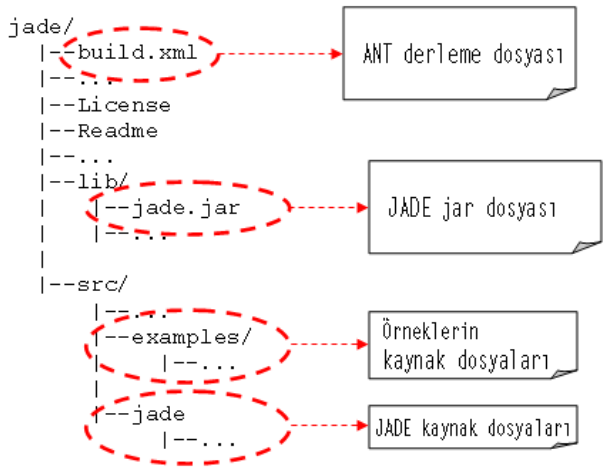

**Şekil 3.10.** Dağıtık paylaşımlı hafıza modeli (Llnl, 2016)

### 3.1.5. Jade çatısı

Jade (Java Agent DEvelopment Framework) Java dilinde yazılmış bir yazılım çatısıdır (framework) ve programcıya temel arakatman işlevlerini sağlayan bir yazılım platformudur. Arakatman; farklı bilgisayarları, ağları ve işletim sistemleri destekleyen ve paralel uygulamalar geliştirmeyi mümkün kılan işletim sistemleri ve uygulamalar arasındaki bir yazılım katmanıdır (Tanenbaum ve Van Steen, 2007). Jade, çoklu-ajan (multi-agent) sistemlerinin kodlamasını ve paralel uygulamaların geliştirilmesini kolaylaştırmak için 1998 yılında Telecom Italy tarafından geliştirilmeye başlanmıştır. Jade 2000 yılından beri açık kaynak (open source) olarak geliştirilmeye devam etmektedir ve LGPL (Library Gnu Public Licence) lisansı altında dağıtılmaktadır (Bellifemine ve ark., 2007). Bu tez çalışmasında 4.2 ve 4.3 versiyonları kullanılmıştır. Şu anki en güncel sürümü ise 4.4 versiyonudur. Jade'nin kullanımı ile ilgili yazılım, örnek kod ve belgeler Jade'nin internet sitesinde (http://jade.tilab.com) bulunmaktadır. Jade İspanyol telekomünikasyon şirketi Telefonica, CNET, Japon medya kuruluşu

NHK, Imperial College London, INRIA ve Helsinki Üniversitesi gibi birçok şirket ve akademik grup tarafından kullanılmaktadır.

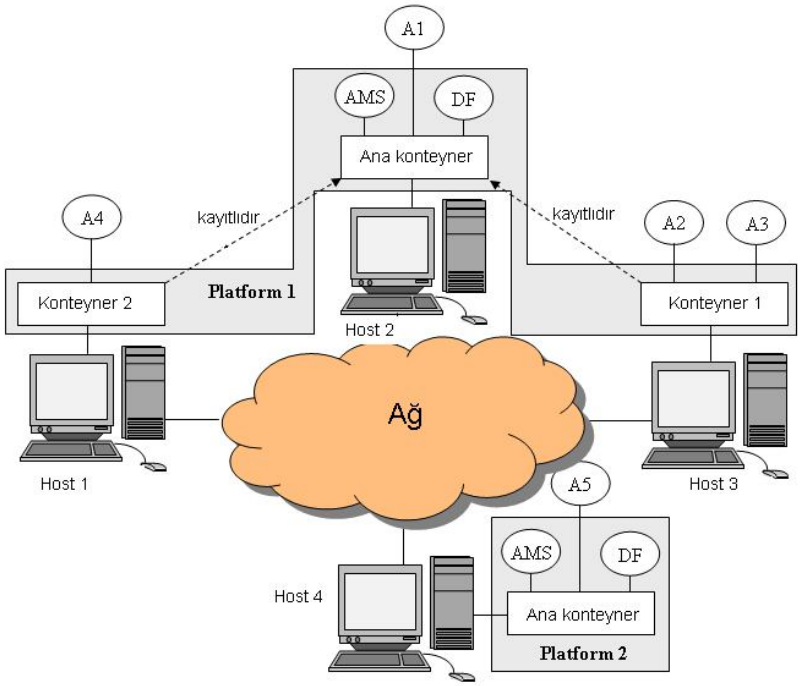
Jade platformu bilgisayarlar arasında dağıtılabilmektedir ve ayarlar uzak bir arayüz ile kontrol edilebilmektedir. Bilgisayarların aynı işletim sistemine sahip olmasına gerek yoktur. Hatta ihtiyaç duyuldukça yeni ajanlar oluşturma ve ajanları bir bilgisayardan diğerine taşıma gibi ayarlar çalışma anında bile yapılabilmektedir. Jade'nin çalışması için sistem gereksinimi olarak sadece Java Runtime 5 veya üstü gerekmektedir. Ayrıca, Jade çatısının proje klasör yapısı Şekil 3.11'de gösterilmektedir.




**Şekil 3.11.** Jade çatısının klasör yapısı (Bellifemine ve ark., 2007)

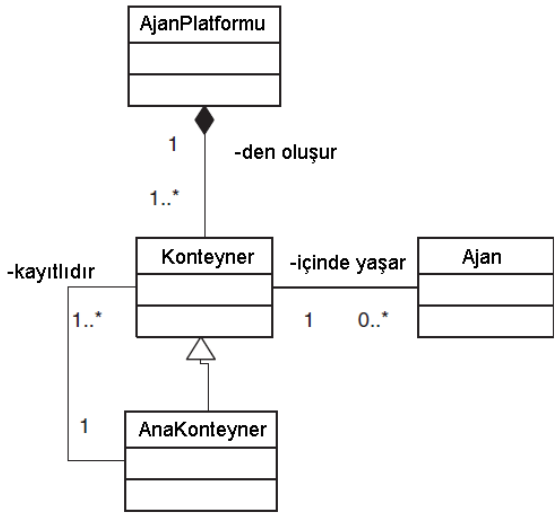
Şekil 3.12'de temel Jade mimari öğeleri gösterilmektedir. Jade tabanlı bir uygulama ajanlar (agents) diye adlandırılan bir dizi bileşenden oluşmaktadır ve her bir ajan benzersiz bir isme sahiptir. Ajanlar kendilerine yüklenen görevleri çalıştırırlar ve mesaj takası ile kendi aralarında iletişime geçerler. Ajanlar kendilerine mesaj iletimi gibi temel servisleri sağlayan bir Platformun üstünde yaşarlar. Şekil 3.13 Jade'nin temel mimari elemanları arasındaki ilişkiyi UML diyagramı olarak göstermektedir. UML diyagramında görüldüğü gibi, bir platform bir ya da daha fazla konteynerden oluşmaktadır. Konteynerler ağ üzerindeki farklı bilgisayarlarda çalıştırılabilir ve böylece bir dağıtık platform elde edilir. Her bir konteyner bir veya birkaç tane ajan içerebilir. Örneğin Şekil 3.12'de, Host 3 bilgisayarındaki "Konteyner 1" isimli konteyneri A2 ve A3 ajanlarını içermektedir. Bir konteyneri bir java sanal makinesi gibi düşünebiliriz. Platformda, "Ana konteyner" isimli özel bir konteyner de bulunmaktadır ve diğer konteynerlerden farklı olarak şu özelliklere sahiptir:

- Platformda başlatılacak ilk konteynerdir ve diğer tüm konteynerler önyükleme anında ana konteynere kayıt olurlar.
- AMS ve DF isminde iki özel ajan içerir. AMS ajanı platformdaki yetkiyi temsil eder ve platformdaki yönetim olaylarını (ajan başlatma veya öldürme, tüm platformu kapatma gibi) gerçekleştirebilen tek ajandır. DF ajanı ise sarı sayfalar (Yellow Pages) servisini sağlar, ajanların sundukları hizmetleri yayınlamaya ve ihtiyaç duyulan hizmetleri sunan ajanları bulmaya yaramaktadır.

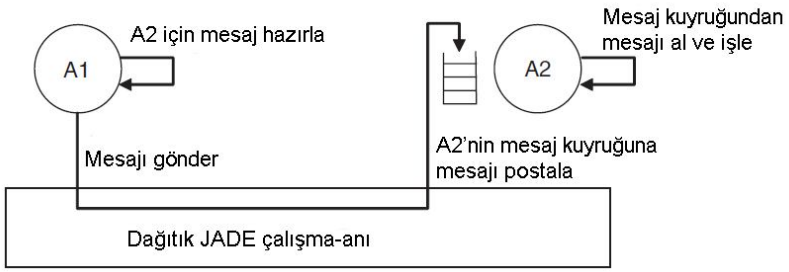



**Şekil 3.12.** Jade mimarisi (Bellifemine ve ark., 2007)

Ajan haberleşmesi Jade'nin en güçlü özelliklerinden bir tanesidir. Ajanlar kendi aralarında aynı konteynerde veya farklı konteynerde yaşadığına bakmaksızın şeffaf ve doğru şekilde haberleşirler. Haberleşme paradigması asenkron mesaj göndermeye dayalıdır. Bu nedenle, her bir ajanın bir posta kutusu (ajan mesaj kuyruğu) vardır. Jade ajanlar tarafından gönderilen mesajları ilgili ajanın posta kutusundaki mesaj kuyruğuna iletmekle sorumludur. Posta kutusundaki mesaj kuyruğuna bir mesaj gönderildiği zaman alıcı ajan bilgilendirilir. Bu aşamada, ajanın kuyruktan aldığı mesajı ne zaman ve nasıl işleyeceğini programcı belirlemektedir. Bu süreç Şekil 3.14'de gösterilmektedir.




**Şekil 3.13.** Temel mimari elemanlar arasındaki ilişki (Bellifemine ve ark., 2007)



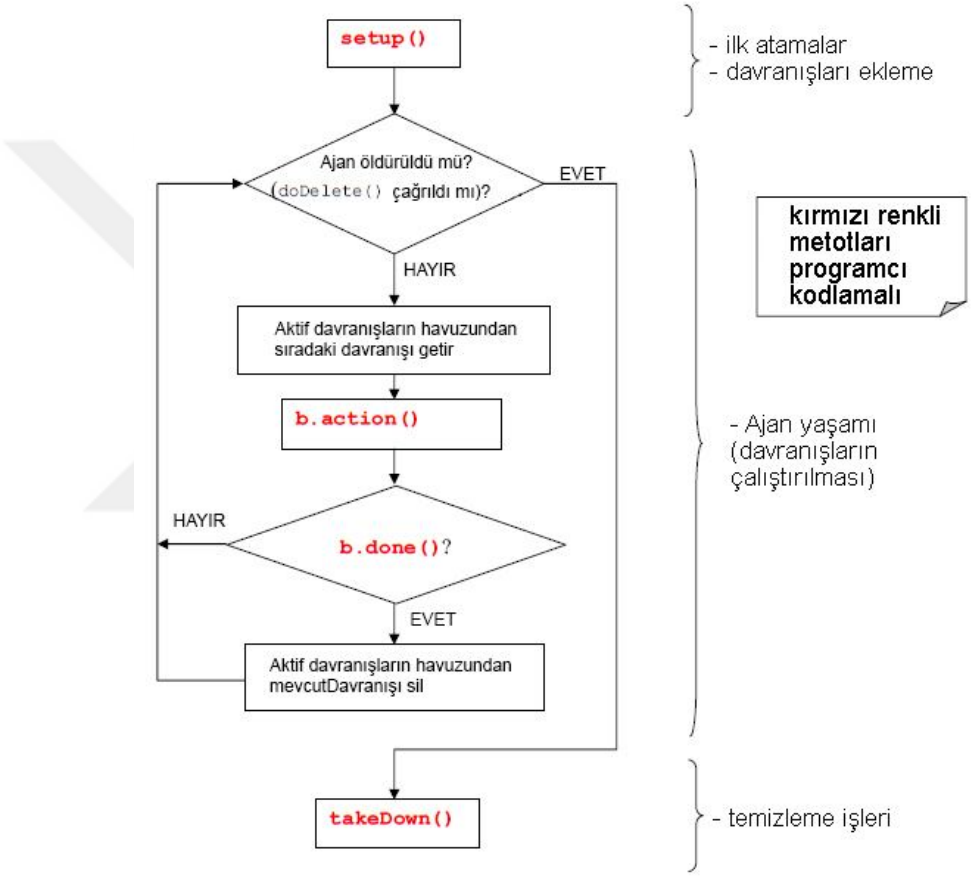

**Şekil 3.14.** Jade asenkron mesaj gönderme örneği (Bellifemine ve ark., 2007)

`jade.core.Agent` sınıfı kullanıcı tanımlı ajanlar için temel sınıfı temsil etmektedir. Bundan dolayı, programcı açısından bakıldığında, Jade ajanı oluşturmak için `Agent` sınıfından türeyen bir sınıf tanımlamak gerekmektedir. Ayrıca bu sınıfın `setup()` metoduna sahip olması gerekmektedir. Aşağıda basit bir örnek verilmiştir:

```
public class MyAgent extends Agent{
      public void setup() {
            System.out.println("Merhaba Dünya. Ben bir ajanım!");
      }
}
```

`setup()` metodu başlangıç değerlerini atama niyetiyle kullanılmaktadır. `setup()` metodu içerisinde ajanın yapması gereken işlemlere örnek olarak arayüz ekranı gösterme, veritabanı bağlantısı kurma ve davranışları başlatma verilebilir. Ajan oluşturma aşamasında Jade çalışma ortamına henüz bir bağ kurulmadığından dolayı ve bunun sonucu olarak `Agent` sınıfından miras alınan bazı metotlar düzgün çalışmayacağı

için yapıcı metot tanımlamak yerine `setup()` metodunu kullanmak ve içinde başlangıç değerlerini atamak tavsiye edilmektedir. `setup()` metodunun çalışıp sona ermesine rağmen ajan hala canlıdır ve bilgisayarın kaynaklarını kullanmaktadır. Ajanı durdurmak için `doDelete()` metodunun çağrılması gerekmektedir. `doDelete()` metodunun çağrılması ajan durmadan önce çeşitli temizlik işlemlerinin gerçekleşmesini sağlayan `takeDown()` metodunun çalışmasını tetiklemektedir. Bu sürecin akış diyagramı Şekil 3.15'de gösterilmektedir.




**Şekil 3.15.** Ajan çalışmasının akış diyagramı (Bellifemine ve ark., 2007)

Ajanın yapması gereken esas işler davranışlar (behaviours) içerisinde gerçekleştirilir. Bir davranış ajan tarafından gerçekleştirilecek olan bir görevi (task) temsil etmektedir ve `jade.core.behaviours.Behaviour`'den türeyen bir sınıfın nesnesi olarak kodlanmaktadır. Ajanın bir görevi yerine getirecek olan davranışı kullanabilmesi için `addBehaviour()` metodu yardımıyla davranış nesnesini eklemesi gerekmektedir. Davranışlar ajan başlarken (`setup()` metodunda) veya diğer davranışların içerisinden herhangi bir zamanda eklenebilmektedir. `Behaviour`

sınıfından türeyen her bir sınıf iki tane metodu içermelidir: `action()` ve `done()`. `action()` metodu davranış çalıştırılırken gerçekleşecek işlemleri belirler. `done()` metodu ise davranışın tamamlanıp tamamlanmadığını belirten bir `boolean` değeri geri döndürür. Eğer davranış tamamlanmışsa tamamlanan bu davranış ajanın çalıştıracağı davranışların havuzundan silinir.

Bir ajan birkaç davranışı eşzamanlı olarak çalıştırabilir. Bir ajandaki davranışların çizelgelemesi (scheduling) işbirlikçidir. Yani, bir davranış çalıştırılmak için planlandığı zaman o davranışın `action()` metodu çağrılır ve metot sona ulaşıncaya kadar çalışır. Bu esnada ajanın başka bir davranışı çalıştırılmaz. Bu nedenle bir ajanın davranışları arasındaki geçişleri programcı belirlemek zorundadır.

Jade'de üç temel davranış tipi mevcuttur: tek-seferlik davranış (one-shot behaviour), döngüsel davranış (cyclic behaviour), jenerik davranış (generic behaviour).

Tek-seferlik davranış sadece bir seferlik döngüye sahiptir ve `action()` metodu bir kere çalışır. `jade.core.behaviours.OneShotBehaviour` sınıfı `true` döndüren `done()` metoduna sahiptir ve aşağıdaki örnekte görüldüğü gibi tek-seferlik davranışlar bu sınıftan türetilmelidir. Bu örnekte, X işlemi sadece bir kere gerçekleşir.

```
public class MyBehaviour extends OneShotBehaviour{
      public void action() {
            // X işlemi
      }
}
```

Döngüsel davranış hiçbir zaman tamamlanmayacak şekilde tasarlanmıştır ve `action()` metodu her çağrıldığında aynı işlemleri çalıştırır. `jade.core.behaviours.CyclicBehaviour` sınıfı `false` döndüren `done()` metoduna sahiptir ve aşağıdaki örnekte görüldüğü gibi yeni döngüsel davranışlar bu sınıftan türetilmelidir. Bu örnekte, ajan davranışı durduruncaya kadar Y işlemi tekrar tekrar gerçekleşir.

```
public class MyBehaviour extends CyclicBehaviour{
      public void action() {
            // Y işlemi
      }
}
```

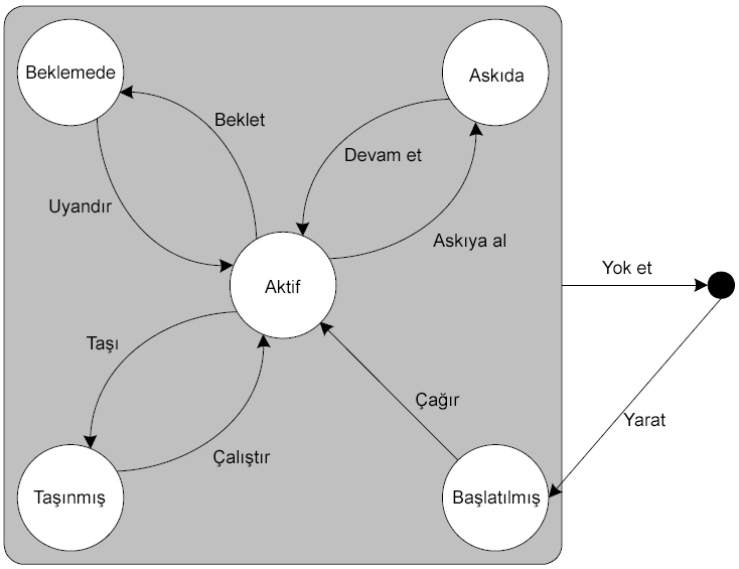
Jenerik davranışlar bir durum değeri tutarlar ve bu durum değerine bağlı olarak farklı işlemleri çalıştırılırlar. Belirli koşul sağlandığında jenerik davranışlar tamamlanır. Aşağıdaki örnekte, `adim` değişkeni davranışın durumunu belirlemektedir. X, Y ve Z işlemleri sırayla çalışırlar.

```
public class MyBehaviour extends Behaviour {
      private int adim = 0;
      public void action() {
            if(adim==0){
                  // X işlemi
            }else if(adim==1){
                  // Y işlemi
            }else if(adim==2){
                  // Z işlemi
            }
            adim++;
      }
      public boolean done() {
            return adim == 3;
      }
}
```

Bir ajan Şekil 3.16'da görüldüğü gibi birkaç durumdan birinde olabilir:

- Başlatılmış: Ajan nesnesi oluşturulmuş, fakat henüz kendini AMS'ye kaydettirmemiş. Bir ismi ve adresi yoktur ve diğer ajanlar ile haberleşemez.
- Aktif: Ajan nesnesi kendini AMS'ye kaydettirir. İsmi ve adresi vardır. Jade'nin sunduğu bütün özellikleri kullanabilir.
- Askıda: Ajan nesnesi durdurulur ve hiçbir ajan davranışı çalıştırılmaz.
- Beklemede: ajan nesnesi durdurulmuştur ve bir mesaj geldiği zaman uyanacaktır.
- Silinmiş: Ajan kesin olarak ölmüştür ve AMS'de artık kayıtlı değildir.
- Taşınmış: Bir ajan yeni bir lokasyona göç ederken bu duruma geçer. Ajanın mesajları geçici hafızada tutulur ve daha sonra yeni lokasyona gönderilir.

`Agent` sınıfı durumlar arasında geçişi sağlayan metotlara sahiptir. Örneğin `doWait()` metodu ajanı aktif durumdan bekleme durumuna geçirir. Dikkat edilmesi gereken bir husus şudur ki bir ajan, davranışlarını ajan sadece aktif durumda iken çalıştırabilir. Eğer herhangi bir davranış `doWait()` metodunu çağıracak olursa tüm ajan ve ajanın bütün aktiviteleri durdurulur. Bunun yerine, `block()` metodu kullanılarak sadece ilgili davranış askıya alınmalıdır.




**Şekil 3.16.** Ajanın yaşam döngüsü (Bellifemine ve ark., 2007)

## 3.2. Kullanılan Optimizasyon Algoritmaları

Bu alt bölümde tez çalışmasında kullanılan parçacık sürü optimizasyonu, kapsamlı öğrenme parçacık sürü optimizasyonu, karınca koloni optimizasyonu ve 3-Opt optimizasyon algoritmaları detaylı olarak anlatılmaktadır.

### 3.2.1. Parçacık sürü optimizasyonu

Bu çalışmanın Giriş bölümünde de bahsedildiği gibi, metasezgisel algoritmalar optimizasyon problemlerini çözmede çok etkilidir. PSO algoritması da bunlardan biridir. Kennedy ve Eberhart (1995) tarafından geliştirilen PSO, popülasyon tabanlı ve metasezgisel bir optimizasyon tekniğidir. PSO algoritması kuş sürülerinin sosyal davranışlarından esinlenmiştir. Popülasyondaki her bir birey *parçacık* olarak adlandırılır ve potansiyel bir çözümü temsil etmektedir. Parçacıklar popülasyondaki mevcut en iyi çözümleri takip ederek arama uzayını taramaktadır ve böylece global optimuma yakınsamaktadır. PSO'nun başarısı ve popülerliği sayesinde PSO, lojistik ve taşımacılık (Wu ve Tan, 2009; Norouzi ve ark., 2017), biyoinformatik (Correa ve ark., 2006; Yuan, 2015), işletme (Yang ve ark., 2011), finans (Kendall ve Su, 2005), veri madenciliği

(Grosan ve ark., 2006; Fong ve ark., 2016), ürün tasarımı ve üretim (Yıldız, 2009), otomotiv endüstrisi (Yildiz, 2012) gibi birçok farklı alanda kullanılmaktadır.

PSO algoritması şu şekilde çalışmaktadır: PSO'daki her bir parçacık bir kuşu temsil etmektedir ve bir çözüm sunmaktadır. Her bir parçacığın uygunluk fonksiyonu ile bulunan bir uygunluk değeri vardır. Parçacıklar arama uzayında kendilerine yön veren hız bilgisine sahiptirler. PSO belirli sayıda rasgele üretilmiş parçacıklar ile başlatılmaktadır. Bu parçacıklar sırasıyla Denklem (3.1) ve (3.2) ile hız ve konum bilgilerini güncelleyerek arama uzayında en uygun sonucu aramaktadırlar. Bir parçacığın konumunu güncellemek için, bu parçacığın önceki en iyi çözümü (*pbest*) ve tüm sürüdeki global en iyi çözüm (*gbest*) kullanılmaktadır. Bu değerler optimizasyon süreci boyunca sürekli güncellenmektedir. Böylece, algoritmanın sonunda global optimum veya global optimuma yakın bir çözüm bulunmaktadır.

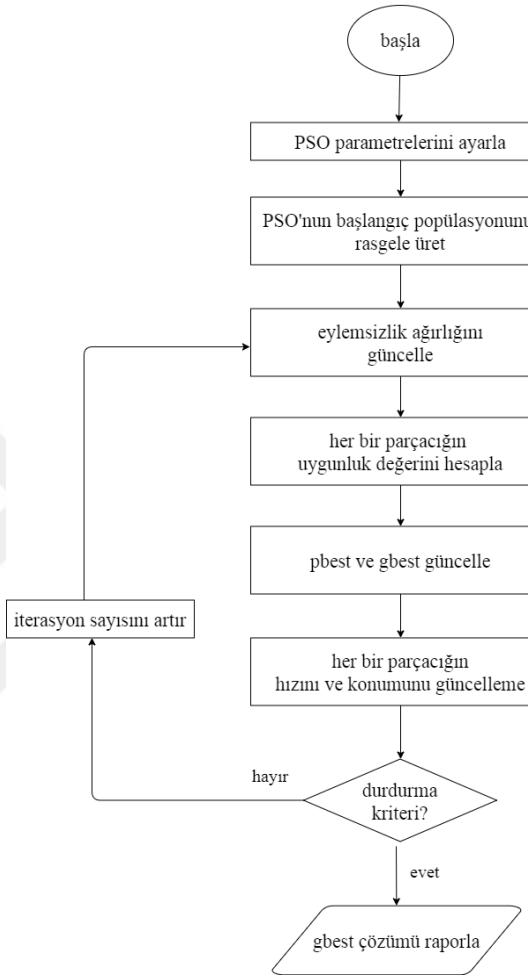
$$V_i^d = w * V_i^d + c_1 * rand1_i^d * (pbest_i^d - X_i^d) + c_2 * rand2_i^d * (gbest^d - X_i^d) \quad (3.1)$$

$$X_i^d = X_i^d + V_i^d \quad (3.2)$$

Burada $V_i^d$ ve $X_i^d$ *i*.nci parçacığın *d*.nci boyutunun sırasıyla hız ve konum bilgisini göstermektedir. İvmelenme katsayısı olan $c_1$ ve $c_2$ bir parçacığın bir iterasyonda alabileceği maksimum adım sayısını etkilemektedir. 0 ile 1 arasındaki rasgele sayılar olan *rand*1 ve *rand*2 algoritmanın stokastik doğasını etkilemektedir (Van Den Bergh, 2006). $pbest_i$ *i*.nci parçacığın ziyaret ettiği en iyi konumdur. *gbest* tüm sürüdeki en iyi konumdur. Eylemsizlik ağırlığı diye adlandırılan *w* sabiti lokal arama kabiliyeti ile global arama kabiliyeti arasındaki dengeyi koruma rolünü oynamaktadır (Shi ve Eberhart, 1998). (Bansal ve ark., 2011)'de görüldüğü gibi birçok eylemsizlik ağırlığı stratejisi önerilmiştir. Shi ve Eberhart (1999)'a göre optimizasyon süreci boyunca eylemsizlik ağırlığı büyük bir değerden küçük bir değere doğrusal olarak azaltılırsa, PSO çalıştırmanın başlangıcında daha fazla global arama yeteneğine sahip olma eğilimindedir ve çalıştırmanın sonuna doğru daha fazla lokal arama yeteneğine sahiptir. Bundan dolayı, eylemsizlik ağırlığı *w* optimizasyon süreci boyunca Denklem (3.3) kullanılarak güncellenmektedir.

$$w(t) = w_{\max} - t \cdot (w_{\max} - w_{\min}) / T \quad (3.3)$$

Burada $w_{max}$ ve $w_{min}$ sırasıyla maksimum ve minimum eylemsizlik ağırlığıdır ve bunlar genellikle 0.9 ve 0.2'ye eşitlenmektedir. $t$ güncel iterasyon sayısıdır ve $T$ ise önceden belirlenmiş maksimum iterasyon sayısıdır. PSO algoritmasının akış diyagramı Şekil 3.17'de gösterilmektedir.




**Şekil 3.17.** PSO algoritmasının akış diyagramı

### 3.2.2. Kapsamlı öğrenme parçacık sürü optimizasyonu

PSO'nun bir türevi olan CLPSO Liang ve ark. (2006) tarafından önerilmiştir. Liang ve ark. PSO'nun esas kusurunun erken yakınsama olduğunu ve ikinci kusurun ise *gbest* parametresinin kullanımı olduğunu belirtmiştir. Denklem (3.1)'de gösterildiği gibi, her bir parçacık *pbest* ve *gbest* değerlerini kullanarak kendi hızını güncellemektedir. *gbest* değeri sürünün tamamını etkilediği için sürünün yönünü belirlemede çok önemlidir. Eğer *gbest* lokal minimuma takılırsa parçacıklarda bu lokal

minimuma kolayca takılabilir. Bu nedenden dolayı, *gbest* CLPSO'da kullanılmamaktadır. Hatta CLPSO'nun bir diğer özelliği, orijinal PSO'daki bir parçacık sadece kendi *pbest'*ini kullanmasına rağmen, CLPSO'daki bir parçacık ise diğer bütün parçacıkların *pbest'*ini de kullanmaktadır. Bu yöntem kapsamlı öğrenme stratejisi olarak adlandırılmaktadır.

Yukarıda bahsedildiği gibi, CLPSO'daki bir parçacığın hızını güncelleme diğer bütün parçacıkların *pbest*'lerine bağlıdır ve bunun formülü Denklem (3.4)'de verilmektedir.

$$V_i^d = w * V_i^d + c * rand_i^d * (pbest_{fi(d)}^d - X_i^d) \tag{3.4}$$

burada $\boldsymbol{f_i}$ = [$f_i$(1), $f_i$(2),…, $f_i$(D)] rasgele seçilmiş parçacıkların bir listesidir. Bu parçacıklar, *i*.nci parçacık dâhil sürüdeki herhangi bir parçacık olabilir. Bunlar Denklem (3.5)'deki öğrenme olasılığı diye adlandırılan *Pc* değeri ile belirlenir. $pbest^d_{fi(d)}$ *d*.nci boyut için *i* parçacığın $\boldsymbol{f_i}$ listesinde kayıtlı olan parçacığın *pbest* değerini göstermektedir. Bir parçacığın her bir boyut için *pbest*leri nasıl seçtiği aşağıda açıklanmıştır ve bu sürecin akış diyagramı Şekil 3.18'de gösterilmiştir:

(1) *rand* sayısı 0 ve 1 arasında rasgele üretilir.

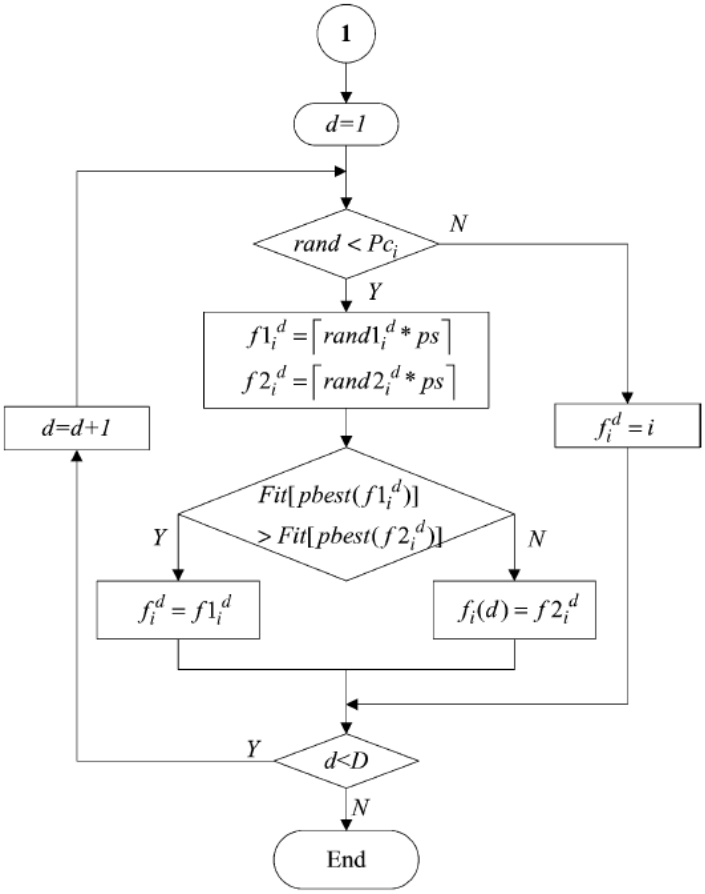
(2) Eğer bu sayı $Pc_i$ değerinden büyükse, *i*. parçacık mevcut boyut için kendi *pbest*'ini kullanmak üzere kendisini $\boldsymbol{f_i}$ listesine ekler ve daha sonra Adım 4'e gidilir. Büyük değilse, turnuva seçimi uygulamak için henüz hızları güncellenmemiş olan iki parçacık seçilir.

(3) Bu seçilen parçacıkların *pbest*lerinin uygunluk değerleri karşılaştırılır. Hangi parçacığın *pbest'*i daha iyi uygunluk değerine sahip ise o parçacık turnuva seçimini kazanır. Kendisinden öğrenilecek bir *örnek* olarak kazananın *pbest'*ini kullanmak için kazanan parçacık mevcut boyut için $\boldsymbol{f_i}$ listesine eklenir.

(4) Eğer maksimum boyut sayısına ulaşılırsa o zaman Adım 5'e gidilir. Değilse, bir sonraki boyuta devam etmek için Adım 1'e gidilir.

(5) Eğer $\boldsymbol{f_i}$ listesindeki bütün örnekler *i*. parçacığın kendisi ise, rasgele bir boyut ve *i*. parçacık hariç rasgele bir parçacık seçilir. Böylece, seçilen boyut için seçilen parçacığın *pbest*'i kullanılır.

$$Pc_i = 0.05 + 0.45 * \frac{\left(\exp\left(\frac{10(i-1)}{ps-1}\right)-1\right)}{(\exp(10)-1)} \tag{3.5}$$




**Şekil 3.18.** *i*. parçacık için örnek boyutların seçimi (*ps*: popülasyon boyutu, ⌈ ⌉: yukarı yuvarlama operatörü) (Liang ve ark., 2006)

PSO ve CLPSO algoritmalarında alt ve üst arama sınırları kullanılmasına rağmen arama sınırlarının kullanımı ile ilgili iki algoritma arasında fark vardır. PSO'da bir parçacığı arama sınırları içerisinde tutmak için $X_i^d$=min($X^d_{max}$, max($X^d_{min}$, $X_i^d$)) eşitliği kullanılmaktadır. Fakat bir parçacığı arama sınırları içerisinde tutmak için CLPSO'da biraz farklı bir metot uygulanmaktadır. Eğer bir parçacık arama sınırları içerisinde bulunuyorsa, o zaman onun uygunluk değerleri hesaplanmaktadır ve onun *pbest* ve *gbest*'i güncellenmektedir. Yani, eğer *i*. parçacığın $\boldsymbol{f_i}$ listesindeki bütün örnekleri arama sınırları içerisinde yer alıyorsa, o zaman bu parçacık arama sınırları içerisine geri dönmüş olmaktadır.

Son olarak, CLPSO'nun bir diğer özelliği boşluk tazeleme diye adlandırılan *m* parametresidir. Boşluk tazeleme minimizasyon problemlerinde lokal minimumlardan

kaçmak ve iyi örneklerden öğrenmek için kullanılmaktadır. Her bir parçacık sahip olduğundan daha iyi örnekler bulmak için araştırma yapmaktadır. Eğer *i*. parçacık, *m* adet iterasyonda daha iyi örnekler bulamazsa, bu parçacık için $\boldsymbol{f_i}$ listesi yeniden atanmaktadır. Liang ve ark. (2006) *m* parametresi değerinin 7 civarında iken daha iyi sonuçlar alınabildiğini bildirmiştir. CLPSO algoritmasının akış diyagramı (Liang ve ark., 2006)'de gösterilmektedir.

### 3.2.3. Karınca koloni optimizasyonu

Dorigo ve ark. tarafından geliştirilen ACO gerçek karıncaların davranışlarından ilham alınan bir metasezgisel algoritmadır (Dorigo ve ark., 1996; Dorigo ve Gambardella, 1997). Gerçek karıncalar bir görme kabiliyetine sahip değildir. Fakat buna rağmen, onlar yuvaları ve yiyecek kaynakları arasındaki en kısa yolu bulabilmektedirler. Karıncalar rasgele gezinmektedirler ve bulunan yiyeceğe bağlı olarak *feromon* diye adlandırılan kimyasal bir maddeyi geçtikleri yollara bırakarak yuvalarına geri dönerler. Diğer karıncalar feromon izini takip eder, yiyecek kaynağını bulur ve yuvaya geri dönerken izi güçlendirir. Zamanla, feromon buharlaşmaya başlar, böylece onun çekici gücü de azalmaya başlar. Yollardaki feromon yoğunluğu seçilecek rotanın belirlenmesinde kullanılmaktadır. Karıncalar feromon yoğunluğunun yüksek olduğu yolları seçerek en uygun kısa yolu bulmaktadırlar.

Karınca koloni optimizasyonu gezgin satıcı problemlerini çözmek için geliştirilmiştir (Dorigo ve Gambardella, 1997) ve ayrıca herhangi bir kombinasyonlu probleme de uygulanabilmektedir. ACO'nun popülerliği ve başarısı sayesinde, ACO veri madenciliği (Wu ve ark., 2012), semantik web (Viswanathan ve Krishnamurthi, 2015), mobil robotların rota planlaması (Liu ve ark., 2016) ve endüstrideki montaj hattı dengeleme (Chica ve ark., 2015) gibi pek çok farklı alanda kullanılmaktadır. GSP *n* şehre ve bir satıcıya sahiptir. Satıcı bütün şehirlere sadece bir kez uğramayı ve minimum uzunluklu kapalı bir tur oluşturmayı hedeflemektedir. Bütün şehirlerarasındaki uzaklıklar bilinmektedir ve $d_{ij}$ *i* ve *j* şehirleri arasındaki uzaklığı temsil etmektedir. Algoritmanın başlangıcında, *m* karınca farklı şehirlere rastgele dağıtılır. *i* şehrindeki *k*. karınca Denklem (3.6)'daki olasılık fonksiyonuna göre *j* şehrine hareket eder.

$$\rho_{ij}^{k} = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}.[\eta_{ij}]^{\beta}}{\sum_{u \in A_k}[\tau_{iu}(t)]^{\alpha}.[\eta_{iu}]^{\beta}} & eger \quad j \in A_k \\ 0 & \deg ilse \end{cases} \tag{3.6}$$

Burada *i*, *j* ve *u* şehirlerdir, $A_k$ ise *k*. karınca tarafından henüz ziyaret edilmemiş şehirlerin listesidir. $\tau_{ij}(t)$; iterasyon *t* anında *i* ve *j* şehirleri arasındaki feromon miktarıdır. $\eta_{ij}$; *i* ve *j* şehirleri arasındaki $1/d_{ij}$ ile verilen sezgisel bilgidir. $\alpha$ ve $\beta$ parametreleri feromon miktarı ve sezgisel bilgi arasındaki göreceli önemi kontrol etmektedir. *k*. karınca bütün şehirleri ziyaret eder ve kapalı bir tur tamamlar. Bütün karıncalar turlarını tamamladıktan sonra global feromon güncelleme kuralı uygulanır. Her bir karınca, üzerinden geçtiği bütün yollara bırakacağı feromon miktarını Denklem (3.7)'ye göre hesaplar.

$$\Delta\tau_{ij}^{\ k} = \frac{Q}{L_k} \tag{3.7}$$

Burada *Q* sabit bir değerdir ve $L_k$; *k*. karıncanın tur uzunluğunu temsil etmektedir. *i* ve *j* şehirleri arasına bütün karıncalar tarafından bırakılacak toplam feromon miktarı Denklem (3.8) ile bulunmaktadır. Daha sonra, buharlaşma süreci meydana gelir. Buharlaşma sürecinden sonra bütün karıncalar üzerinden geçtikleri yollara feromonları bırakırlar. Buharlaşma ve feromon bırakma olayı Denklem (3.9)'a göre gerçekleştirilir.

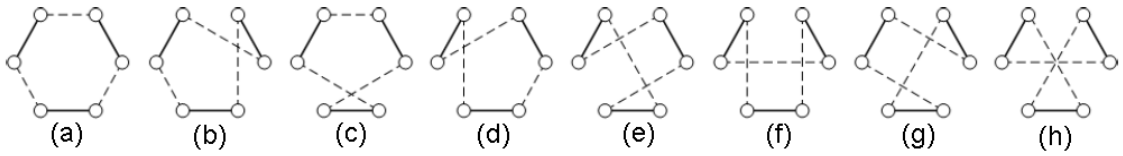
$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{\ k} \tag{3.8}$$

$$\tau_{ij}(t+1) = (1-\rho).\tau_{ij}(t) + \Delta\tau_{ij} \tag{3.9}$$

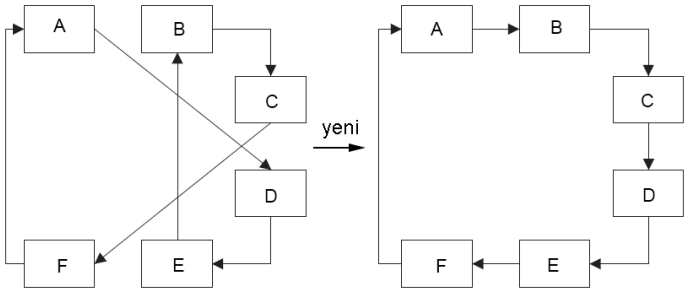
Denklem (3.9)'da, $\rho$ feromon buharlaşma katsayısıdır ve 0 ile 1 arasında bir değer alır. Böylece, bir iterasyon tamamlanır ve algoritma bir sonraki iterasyona başlar. Algoritma maksimum iterasyon sayısı gibi bir durdurma şartına kadar devam eder. Algoritmanın sonunda, en kısa tur uzunluğu elde edilir.

### 3.2.4. 3-Opt lokal arama algoritması

GSP için birkaç tane tur iyileştirme algoritması vardır. Bunlardan bir tanesi de *k*-Opt algoritmasının bir alt sınıfı olan 3-Opt lokal arama algoritmasıdır. 3-Opt algoritması bir turdaki üç kenarı siler, böylece turu üç parçaya ayırır ve daha sonra bu parçaları bütün olasılıkları deneyerek tekrar birleştirir ve en iyi olanını seçer. 3-Opt algoritması iyileştirme bulunamayıncaya kadar turu silmeye ve yeniden oluşturmaya devam eder (Johnson ve McGeoch, 1997). Böylece, tur artık 3-optimaldir. 3-Opt'daki üç kenarın silinmesiyle, Şekil 3.19'da görüldüğü gibi tekrar birleştirme için olası sekiz durum bulunmaktadır. Şekil 3.20 ise 3-Opt sezgisel hareketin bir örneğini vermektedir. Şekil 3.20'de görüldüğü gibi, sol taraftaki turda üst üste binmiş kenarlar mevcuttur. Sağ tarafta, 3-Opt lokal arama algoritması sayesinde daha kısa tur uzunluğuna sahip yeniden oluşturulmuş tur vardır. Birçok meta sezgisel algoritma GSP'nin çözümü için kullanılmaktadır. Lokal arama algoritmaları çözümleri yerel olarak iyileştirdiği için ACO algoritması lokal arama algoritmalarıyla birleştirildiği zaman çok iyi bir performans elde etmektedir (Dorigo ve Stützle, 2004).




**Şekil 3.19.** Bütün olası 3-Opt yeniden bağlama durumu




**Şekil 3.20.** Olası bir 3-Opt yeniden bağlama durumu

### 3.3. Tek Amaçlı Optimizasyon Problemleri

Eğer problemde optimize edilecek tek bir amaç var ise bu problem *tek amaçlı optimizasyon problemi* olarak tanımlanmaktadır. Tek amaçlı optimizasyonda, tüm farklı hedefleri bir arada toplayan tek bir hedef fonksiyonun minimum veya maksimum değerine karşılık gelen en iyi çözümü bulmak amaçlanmaktadır (Savic, 2002). Aşağıda tez çalışmasında kullanılan tek amaçlı optimizasyon problemleri sunulmaktadır.

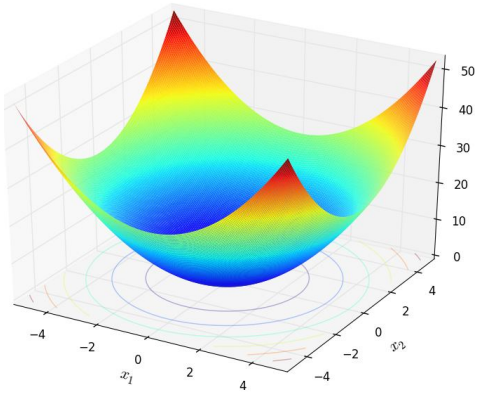
#### 3.3.1. Kıyaslama fonksiyonları

PCLPSO algoritmasının performansını test etmek ve diğer çalışmalarla karşılaştırmak için 2 adet tek modlu (tek optimum nokta içeren) ve 12 adet çok modlu (birçok yerel minimum ancak bir genel minimum içeren) kıyaslama fonksiyonları seçilmiştir ve deneylerde kullanılmıştır. Bu fonksiyonlar global optimizasyon üzerine çalışanlar tarafından iyi bilinmektedirler ve optimizasyon algoritmalarının testi ve analizi için sıklıkla kullanılmaktadırlar. Bu 14 fonksiyonun 2B model grafiği Şekil 3.21'de gösterilmektedir. Formülleri aşağıda verilmektedir ve özellikleri aşağıda detaylı olarak anlatılmaktadır. Bu 14 fonksiyonun global optimum değerleri, ilk değer alma aralığı ve arama aralığı Çizelge 3.1'de sunulmaktadır. Tez çalışmasında kullanılan bu 14 fonksiyon Denklem 3.10 ile Denklem 3.24 arasında verilmiştir.

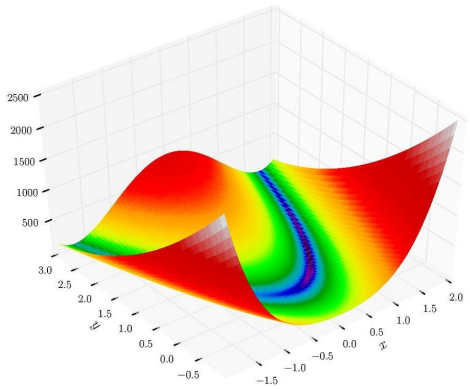
**Çizelge 3.1.** Fonksiyonların tipi, Global minimumu, Fonksiyon değeri, Arama ve İlk değer aralığı

| $f$ | Tip | Global minimum $x^*$ | Fonksiyon değeri $f(x^*)$ | Arama aralığı | İlk değer aralığı |
|---|---|---|---|---|---|
| $f_1$ | T | [0,0,…,0] | 0 | $[-100, 100]^D$ | $[-100, 50]^D$ |
| $f_2$ | T | [1,1,…,1] | 0 | $[-2.048, 2.048]^D$ | $[-2.048, 2.048]^D$ |
| $f_3$ | Ç | [0,0,…,0] | 0 | $[-32.768, 32.768]^D$ | $[-32.768, 16]^D$ |
| $f_4$ | Ç | [0,0,…,0] | 0 | $[-600, 600]^D$ | $[-600, 200]^D$ |
| $f_5$ | Ç | [0,0,…,0] | 0 | $[-0.5, 0.5]^D$ | $[-0.5, 0.2]^D$ |
| $f_6$ | Ç | [0,0,…,0] | 0 | $[-5.12, 5.12]^D$ | $[-5.12, 2]^D$ |
| $f_7$ | Ç | [0,0,…,0] | 0 | $[-5.12, 5.12]^D$ | $[-5.12, 2]^D$ |
| $f_8$ | Ç | [420.96,420.96,…,420.96] | 0 | $[-500, 500]^D$ | $[-500, 500]^D$ |
| $f_9$ | DÇ | [0,0,…,0] | 0 | $[-32.768, 32.768]^D$ | $[-32.768, 16]^D$ |
| $f_{10}$ | DÇ | [0,0,…,0] | 0 | $[-600, 600]^D$ | $[-600, 200]^D$ |
| $f_{11}$ | DÇ | [0,0,…,0] | 0 | $[-0.5, 0.5]^D$ | $[-0.5, 0.2]^D$ |
| $f_{12}$ | DÇ | [0,0,…,0] | 0 | $[-5.12, 5.12]^D$ | $[-5.12, 2]^D$ |
| $f_{13}$ | DÇ | [0,0,…,0] | 0 | $[-5.12, 5.12]^D$ | $[-5.12, 2]^D$ |
| $f_{14}$ | DÇ | [420.96,420.96,…,420.96] | 0 | $[-500, 500]^D$ | $[-5.12, 5.12]^D$ |

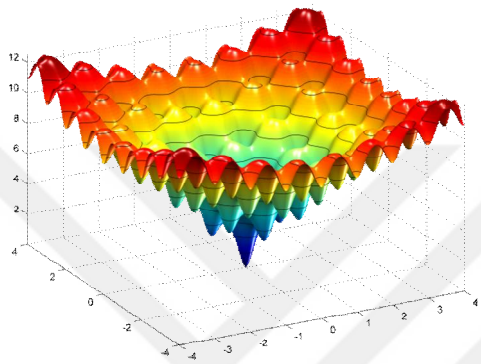
(T: Tek modlu, Ç: Çok modlu, DÇ: Döndürülmüş çok modlu)



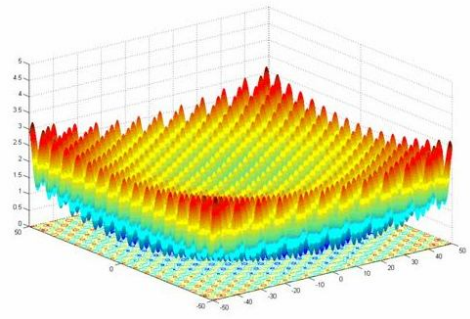
a) Sphere



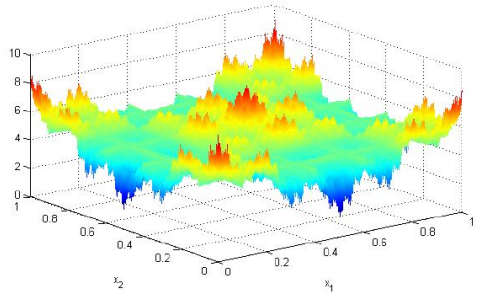
b) Rosenbrock



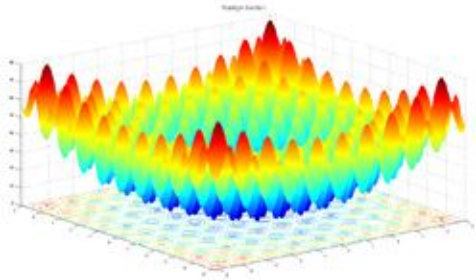
c) Ackley



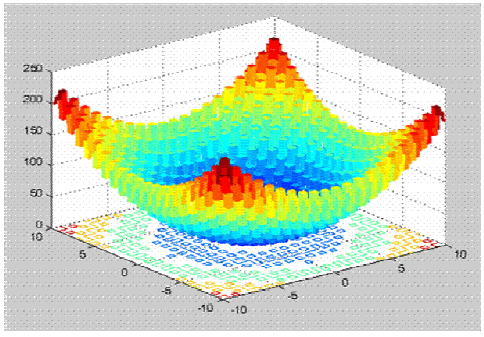
d) Griewank



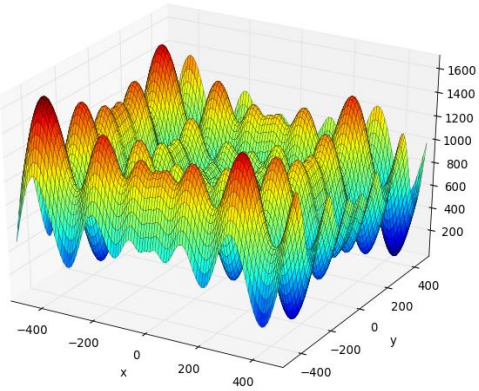
e) Weierstrass



f) Rastrigin



g) Noncontinuous Rastrigin



h) Schwefel

**Şekil 3.21.** Kıyaslama fonksiyonlarının 2B model grafiği

Sphere fonksiyonu:

$$f_1(x) = \sum_{i=1}^{D} x_i^2 \tag{3.10}$$

Rosenbrock fonksiyonu:

$$f_2(x) = \sum_{i=1}^{D-1} [100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2] \tag{3.11}$$

$f_1$ ve $f_2$ fonksiyonları tek modludur. Tek modlu fonksiyonlar sadece bir tane optimuma sahiptir ve hiç lokal minimumu yoktur. $f_1$ fonksiyonu birçok araştırmacı tarafından araştırmalarda kullanılmış olan Sphere fonksiyonudur. Sphere fonksiyonunun çözümü kolay olarak bilinmektedir. $f_2$ fonksiyonu Rosenbrock fonksiyonudur. Klasik Rosenbrock fonksiyonu iki boyutlu tek modlu fonksiyondur. Son yıllarda daha yüksek boyutlara genişletilmiştir. Birçok araştırmacı yüksek boyutlu Rosenbrock fonksiyonunu tek modlu fonksiyon olarak kabul etmektedir. Fakat Shang ve Qiu (2006) *n*-boyutlu (*n*=4~30) Rosenbrock fonksiyonunun 2 tane minimum değere sahip olduğunu göstermektedirler. Bundan dolayı $f_2$ çok modlu fonksiyon olarak da kabul edilmektedir.

Ackley fonksiyonu:

$$f_3(x) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right) + 20 + e \tag{3.12}$$

Griewank fonksiyonu:

$$f_4(x) = \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{3.13}$$

Weierstrass fonksiyonu:

$$f_5(x) = \sum_{i=1}^{D}\left(\sum_{k=0}^{k\max}\left(a^k \cos\left(2\pi b^k (x_i + 0.5)\right)\right)\right) - D\sum_{k=0}^{k\max}\left(a^k \cos\left(2\pi b^k * 0.5\right)\right) \tag{3.14}$$

$a$=0.5, $b$=3, $k_{\max}$ = 20.

Rastrigin fonksiyonu:

$$f_6(x) = 10D + \sum_{i=1}^{D} \left[ x_i^2 - 10\cos(2\pi x_i) \right] \tag{3.15}$$

Noncontinuous Rastrigin fonksiyonu:

$$f_7(x) = 10D + \sum_{i=1}^{D} \left[ y_i^2 - 10\cos(2\pi y_i) \right]$$

$$y_i = \begin{cases} x_i & |x_i| < \frac{1}{2} \\ \frac{round(2x_i)}{2} & |x_i| >= \frac{1}{2} \end{cases} \quad for\ i = 1,2,...,D. \tag{3.16}$$

Schwefel fonksiyonu:

$$f_8(x) = 418.9829D - \sum_{i=1}^{D} x_i \sin\left(\sqrt{|x_i|}\right) \tag{3.17}$$

$f_3$ - $f_8$ fonksiyonları çok modludur. Çok modlu fonksiyonlar sadece bir tane optimuma ve birçok lokal minimuma sahiptir. Çok modlu fonksiyonlar araştırmacılar tarafından kıyaslama fonksiyonlarının zor bir sınıfı olarak kabul edilir. Çünkü fonksiyonun boyut sayısı arttıkça bu fonksiyonun lokal minimum sayısı üstel olarak artmaktadır (Yao ve ark., 1999; Esquivel ve Coello Coello, 2003; Li, 2010; Qu ve ark., 2013). Bu nedenle, çok modlu fonksiyonlar üzerinde iyi sonuçlar elde etmek optimizasyon algoritmaları için çok önemlidir. Çünkü bu iyi sonuçlar algoritmanın lokal optimumlardan ve platolardan kaçtığını ve global optimuma doğru hareket ettiğini göstermektedir. Dahası, optimizasyon algoritmaları çok modlu fonksiyonları çözerken lokal minimuma kolayca düşebilmektedir. Bundan dolayı, global arama kabiliyetine (exploration) sahip olan optimizasyon algoritmaları büyük olasılıkla daha iyi sonuçlar elde edebilmektedir.

$f_3$ fonksiyonu; problem uzayının kenarlarına yakın düz bir alana ve problem uzayının merkezinde geniş bir boşluğa sahip olan Ackley fonksiyonudur. $f_3$'ün lokal minimumları birinci üstel ifadenin baskınlığından dolayı büyük boyutlar için sığdır. Bu nedenle $f_3$, $f_3$ - $f_8$ arasındaki en kolay problem olarak kabul edilebilmektedir. $f_4$ birçok lokal minimuma ve sığ bir eğriye sahip olan Griewank fonksiyonudur. Bu bir çelişkidir ki düşük boyutlu Griewank fonksiyonunu çözmek yüksek boyutludan daha zordur (Chen ve ark., 2011). $f_5$ sürekli olan fakat sadece noktalar kümesinde diferansiyeli

alınabilen Weierstrass fonksiyonudur. $f_6$ çok sayıda lokal minimuma sahip olduğu için optimizasyon algoritmalarının performansını test etmek için sıklıkla kullanılan Rastrigin fonksiyonudur. $f_7$ Noncontinuous Rastrigin fonksiyonudur ve $f_6$ temellidir. $f_6$'nın lokal minimum sayısı $f_6$ ile aynı boyut sayısına sahip $f_7$'nin lokal minimum sayısına eşittir. $f_8$ Schwefel fonksiyonudur. $f_8$'in global optimumunu bulmak çok zordur. Çünkü onun lokal optimumları derindir ve global optimuma en yakın lokal optimum global optimumdan oldukça uzaktır.

$f_9$ - $f_{14}$ fonksiyonları $f_3$ - $f_8$'in döndürülmüş versiyonlarıdır. Döndürme işlemi, ayrılabilir fonksiyonları daha zor çözülen ayrılamayan fonksiyonlara çevirmektedir. Ayrılabilir bir fonksiyon Denklem (3.18) kullanılarak döndürülür. Formüldeki **M** matrisi dikey matrisi temsil etmektedir (Salomon, 1996) ve **y** değişkeni fonksiyonun yeni giriş vektörünü temsil etmektedir. Dikey matris ne fonksiyonun şeklini ne de global çözümü değiştirmez.

$$\mathbf{y} = \mathbf{M} * \mathbf{x} \tag{3.18}$$

Rotated Ackley fonksiyonu:

$$f_9(x) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} y_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi y_i)\right) + 20 + e \tag{3.19}$$

burada $\boldsymbol{y} = \mathbf{M} * \boldsymbol{x}$.

Rotated Griewank fonksiyonu:

$$f_{10}(x) = \sum_{i=1}^{D}\frac{y_i^2}{4000} - \prod_{i=1}^{D}\cos\left(\frac{y_i}{\sqrt{i}}\right) + 1 \tag{3.20}$$

burada $\boldsymbol{y} = \mathbf{M} * \boldsymbol{x}$.

Rotated Weierstrass fonksiyonu:

$$f_{11}(x) = \sum_{i=1}^{D}\left(\sum_{k=0}^{k\max}\left(a^k \cos\left(2\pi b^k (y_i + 0.5)\right)\right)\right) - D\sum_{k=0}^{k\max}\left(a^k \cos\left(2\pi b^k * 0.5\right)\right) \tag{3.21}$$

burada $a$=0.5, $b$=3, $k_{\max}$ = 20, $\boldsymbol{y} = \mathbf{M} * \boldsymbol{x}$.

Rotated Rastrigin fonksiyonu:

$$f_{12}(x) = 10D + \sum_{i=1}^{D}\left[y_i^2 - 10\cos(2\pi y_i)\right] \tag{3.22}$$

burada $\boldsymbol{y} = \mathbf{M} * \boldsymbol{x}$.

Rotated Noncontinuous Rastrigin fonksiyonu:

$$f_{13}(x) = 10D + \sum_{i=1}^{D}\left[z_i^2 - 10\cos(2\pi * z_i)\right]$$

$$z_i = \begin{cases} y_i & if\,|y_i| < \frac{1}{2} \\ \frac{round(2y_i)}{2} & if\,|y_i| >= \frac{1}{2} \end{cases} \quad for\ i = 1,...,D. \tag{3.23}$$

burada $\boldsymbol{y} = \mathbf{M} * \boldsymbol{x}$.

Rotated Schwefel fonksiyonu:

$$f_{14}(x) = 418.9829 * D - \sum_{i=1}^{D} z_i$$

$$z_i = \begin{cases} y_i \sin\left(\sqrt{|yi|}\right) & if\,|y_i| <= 500 \\ 10^{-3} * \left(|y_i| - 500\right)^2 & if\,|y_i| > 500 \end{cases} \quad for\ i = 1,...,D. \tag{3.24}$$

burada $\boldsymbol{y} = \boldsymbol{y'} + 420.96$, $\boldsymbol{y'} = \mathbf{M} * (\boldsymbol{x} - 420.96)$

### 3.3.2. Gezgin satıcı problemi

Hesaplamalı karmaşıklık teorisi bilgisayar bilimi ve matematikte temel bir daldır. Hesaplamalı karmaşıklık teorisi, hesaplamalı problemlerin zorluk derecesine göre hesaplamalı problemleri sınıflandırmaya odaklanmaktadır. Bu problemlerden bir tanesi de kombinasyonlu optimizasyon problemi olan gezgin satıcı problemidir (GSP). GSP'de, bir gezgin satıcı bütün şehirlere sadece bir kez uğramayı ve başladığı şehire geri dönerek kapalı bir tur oluşturmayı amaçlamaktadır. Bu tur, gezilen yolun uzunluğunu temsil etmektedir. Şehir sayısı arttıkça optimum tur uzunluğunu belirlemek çok zorlaşmaktadır. Bu nedenle bir GSP karar problemi genellikle NP-tam problemi olarak sınıflandırılmaktadır (Donald, 2011). Metasezgisel algoritmalar genellikle bir çözüm havuzu üzerinde çalışır ve en uygun çözümü elde etmeyi amaçlamaktadır. Bu nedenle, çözülen bu problemler optimizasyon problemleri olarak sınıflandırılır. GSP, bir

optimizasyon problemi olarak çözüldüğünde NP-zor problem olarak sınıflandırılmaktadır (Donald, 2011).

GSP'nin birkaç çeşidi bulunmaktadır ve bunlardan bir tanesi *simetrik* GSP'dir. Simetrik GSP'nin tanımı şu şekildedir: *n* düğümlü bir küme ve düğümlerin her bir çifti için uzaklıklar göz önüne alındığında, amaç her düğümü bir kereliğine ziyaret eden minimum toplam uzunlukta bir tur bulmaktır. Düğüm *i*'den düğüm *j*'ye olan uzaklık düğüm *j*'den düğüm *i*'ye olan uzaklık ile aynıdır (Reinelt, 1995).

Bu tez çalışmasında simetrik GSP'nin çözümü gerçekleştirilmiştir. Deneylerde kullanılan problem örnekleri ve özellikleri Çizelge 3.2'de gösterilmektedir. Bu problem örnekleri, GSP kıyaslama örneklerinin herkesçe ulaşılabilen koleksiyonu olan TSPLIB (Reinelt, 1991; 1995) kütüphanesinden alınmıştır. Bu örnekler global optimizasyon topluluğu tarafından iyi bilinmektedir ve optimizasyon algoritmalarının testi için yaygın olarak kullanılırlar. GSP örneğinin isminin sonundaki sayı, problemin boyutunu belirtmektedir. Örneğin eil51, 51 şehirli bir problemdir. Şehir sayısı arttıkça optimal turu belirlemek oldukça zorlaşmaktadır.

**Çizelge 3.2.** Deneylerde kullanılan GSP örnekleri ve özellikleri

| Problem adı | Şehir sayısı | Bilinen en iyi sonuç |
|---|---|---|
| eil51 | 51 | 426 |
| berlin52 | 52 | 7542 |
| rat99 | 99 | 1211 |
| eil76 | 76 | 538 |
| st70 | 70 | 675 |
| kroA100 | 100 | 21282 |
| lin105 | 105 | 14379 |
| kroA200 | 200 | 29368 |
| ch150 | 150 | 6528 |
| eil101 | 101 | 629 |
| rd400 | 400 | 15281 |
| fl417 | 417 | 11861 |
| pr439 | 439 | 107217 |
| pcb442 | 442 | 50778 |
| d493 | 493 | 35002 |
| u574 | 574 | 36905 |
| rat575 | 575 | 6773 |
| p654 | 654 | 34643 |
| d657 | 657 | 48912 |
| u724 | 724 | 41910 |
| rat783 | 783 | 8806 |

Amaç fonksiyonu Denklem (3.25) ile gösterilmektedir.

$$\text{minimize} \quad \sum_{i,j} d_{ij} x_{ij} \qquad i \neq j \tag{3.25}$$

Burada $d_{ij}$ $i$ ve $j$ şehirleri arasındaki mesafeyi temsil etmektedir. $x_{ij}$ ise $i$ ve $j$ şehirlerinin arasındaki yolun turda kullanılıp kullanılmadığını belirlemektedir. Eğer $i$ ve $j$ şehirlerinin arasındaki yol turda kullanılmış ise $x_{ij}$ değeri 1 olarak, kullanılmamış ise $x_{ij}$ değeri 0 olarak atanmaktadır (Orman ve Williams, 2006).

### 3.4. Wilcoxon Sıra Toplam Testi

Wilcoxon sıra toplam testi (Wilcoxon rank sum test) iki bağımsız basit tesadüfi örneğe dayandırılır ve bu örneklerin alınmış olduğu anakütlelerin karşılaştırılması amacıyla kullanılır (Johnson ve Bhattacharyya, 2014). Wilcoxon sıra toplam testi anakütleler arasında anlamlı farklılıkların olup olmadığını tespit etmeyi amaçlamaktadır. Sıfır hipotezi bu iki anakütlenin arasında farklılık olmadığı şeklinde kurulur. Wilcoxon sıra toplam testinin test istatistiği ve kritik değerler örnek büyüklüklerine göre farklı değerler almaktadır (Ross, 2014). Bundan dolayı Wilcoxon sıra toplam testi küçük örnekler ve büyük örnekler için farklı şekilde yapılmaktadır. Örnek büyüklükleri $n_1$ ve $n_2$ olmak üzere $n_1 \leq 10$ ve $n_2 \leq 10$ durumunda küçük örnek testi uygulanır. Testin adımları aşağıdaki şekildedir (Kartal, 2014):

**1. Adım**: Hipotez kurulumu

$H_0$: incelenen özellik açısından iki anakütle arasında farklılık yoktur.

$H_1$: incelenen özellik açısından anakütleler birbirinden farklıdır (çift yönlü test). veya

$H_1$: 1. anakütlenin değerleri daha büyüktür (sol kuyruk testi). veya

$H_1$: 2. anakütlenin değerleri daha büyüktür (sağ kuyruk testi).

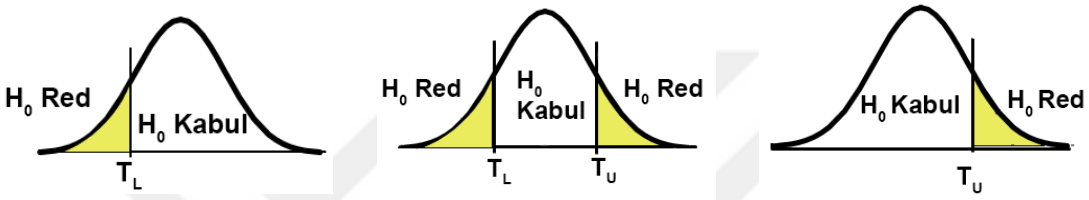
**2. Adım**: Test istatistiği

Wilcoxon sıra toplam testinde test istatistiğini ($W$) hesaplamak için her iki örneğin verileri birlikte küçükten büyüğe sıralanır ve bunlara 1'den başlayarak ardışık olarak sıra numarası verilir. Yani, en küçük verinin sıra numarası 1 ve en büyük verinin sıra numarası $n_1+n_2$ olur. 1. örneğin sıraları toplamı $T_1$ ve 2. örneğin sıraları toplamı $T_2$ olmak üzere test istatistiği Denklem (3.26) ile belirlenmektedir.

$$W = \begin{cases} T_1 \ ve \ T_2 \ den \ küçük \ olanı \ (sol \ kuyruk \ testi \ için) \\ T_1 \ ve \ T_2 \ den \ büyük \ olanı \ (sağ \ kuyruk \ testi \ için) \\ T_1 \ ve \ T_2 \ den \ herhangi \ birisi \ (çift \ yönlü \ test \ için) \end{cases} \quad (3.26)$$

**3. Adım**: Karar modeli

Kritik değerler, Çizelge EK-1.1 ve Çizelge EK-1.2'de gösterilen Wilcoxon sıra toplam testi kritik değerler çizelgesi kullanılarak belirlenmektedir. Testin yönüne göre karar modelleri Şekil 3.22'deki gibidir. $T_L$ ve $T_U$ çizelgedeki kritik değerleri temsil etmektedir.




a) Sol kuyruk testi b) Çift yönlü test c) Sağ kuyruk testi

**Şekil 3.22.** Wilcoxon sıra toplam testi karar modeli

**4. Adım**: Karar

Hesaplanan test istatistiği ($W$) kabul bölgesine denk gelirse $H_0$ hipotezi kabul edilir ve bu iki anakütlenin aynı olduğuna $\alpha$ önem seviyesinde karar verilir. Hesaplanan test istatistiği ($W$) red bölgesine denk gelirse $H_0$ hipotezi reddedilir ve bu iki anakütlenin farklı olduğuna $\alpha$ önem seviyesinde karar verilir.

Örnek büyüklüklerinden birisi veya ikisi 10'u geçtiği durumlarda büyük örnek testi uygulanır. Bu durumda, testin adımları aşağıdaki şekildedir:

**1. Adım**: Hipotez kurulumu

Küçük örnek testindeki ile aynıdır.

**2. Adım**: Test istatistiği

Küçük örnek testindeki adım uygulanır. Daha sonra, Denklem (3.27), (3.28) ve (3.29) ile $W$'nin ortalaması, standart sapması ve $Z$ değeri elde edilir.

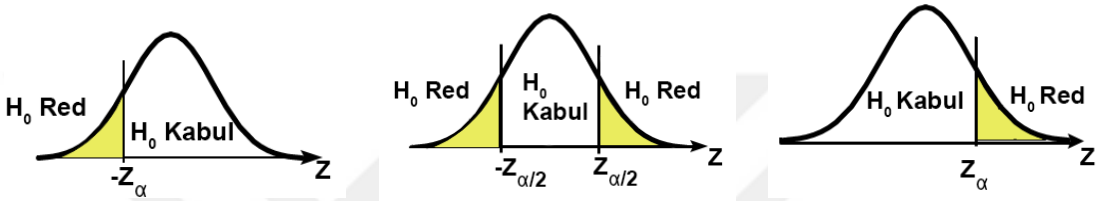
$$\mu_w = \frac{n_1(n_1 + n_2 + 1)}{2} \quad (3.27)$$

$$\sigma_w = \sqrt{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}} \quad (3.28)$$

$$Z = \frac{W - \mu_w}{\sigma_w} \tag{3.29}$$

**3. Adım**: Karar modeli

Çift yönlü test için kritik değerler, Çizelge EK-1.3'deki $0.5 - \left(\alpha/2\right)$'ye karşılık gelen en yakın $Z$ değeridir. Sağ ve sol kuyruk testleri için kritik değerler, Çizelge EK-1.3'deki $0.5 - \alpha$ 'ya karşılık gelen en yakın $Z$ değeridir. Grafiksel olarak Şekil 3.23'de gösterilmektedir.




a) Sol kuyruk testi b) Çift yönlü test c) Sağ kuyruk testi

**Şekil 3.23.** Z tablosu kullanılarak Wilcoxon sıra toplam testi karar modeli

**4. Adım**: Karar

Hesaplanan $Z$ değeri kabul bölgesine denk gelirse $H_0$ hipotezi kabul edilir ve bu iki anakütlenin aynı olduğuna $\alpha$ önem seviyesinde karar verilir. Hesaplanan $Z$ değeri red bölgesine denk gelirse $H_0$ hipotezi reddedilir ve $H_1$ hipotezi kabul edilir. Böylece, bu iki anakütlenin farklı olduğuna $\alpha$ önem seviyesinde karar verilir.

Wilcoxon sıra toplam testinin kullanımını bir örnek ile gösterelim. CLPSO algoritması ile bu tez çalışmasında geliştirilen PCLPSO algoritması 100 boyutlu Sphere test fonksiyonu üzerinde bağımsız olarak 30 defa çalıştırılmıştır. Elde edilen sonuçlar Çizelge 3.3'de gösterilmektedir. Bu sonuçlara göre bu iki algoritma arasında 95% ($\alpha=0.05$) önem seviyesinde istatistiksel olarak anlamlı bir fark olup olmadığı, örnek büyüklükleri ($n_1$=30, $n_2$=30) 10'dan büyük olduğu için büyük örnek testi uygulanarak belirlenmektedir. Testin adımları aşağıdaki şekildedir.

**1. Adım**: Hipotez kurulumu

$H_0$: CLPSO ve PCLPSO algoritmaları arasında farklılık yoktur.

$H_1$: CLPSO ve PCLPSO algoritmaları birbirinden farklıdır. (çift yönlü test)

**2. Adım**: Test istatistiği

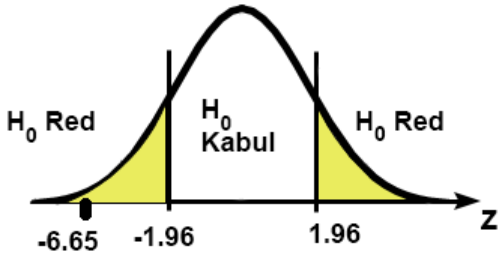
Her iki algoritmaların sonuçları küçükten büyüğe sıralanır ve bunlara 1'den başlayarak ardışık olarak sıra numarası verilir. Sonuçların sıra numaralı verilmiş hali Çizelge 3.3'de gösterilmektedir. CLPSO algoritmasının sıraları toplamı $T_{CLPSO}$ =1365 ve PCLPSO algoritmasının sıraları toplamı $T_{PCLPSO}$ =465'dir. Denklem (3.26)'a göre W değerini $W$= $T_{PCLPSO}$ =465 olarak alınırsa Denklem (3.27)'e göre $\mu_w$= 915, Denklem (3.28)'e göre $\sigma_w$= 67.64, Denklem (3.29)'e göre $Z$ = -6.65 olarak hesaplanmaktadır.

**Çizelge 3.3.** Wilcoxon sıra toplam testi örneği

| # | CLPSO | PCLPSO | Sıra(CLPSO) | Sıra(PCLPSO) |
|---|---|---|---|---|
| 1 | 1.38E-04 | 1.03E-08 | 44 | 29 |
| 2 | 1.80E-04 | 9.61E-09 | 58 | 27 |
| 3 | 1.62E-04 | 3.21E-09 | 54 | 11 |
| 4 | 1.14E-04 | 3.22E-09 | 36 | 12 |
| 5 | 1.49E-04 | 5.70E-09 | 51 | 17 |
| 6 | 1.63E-04 | 1.34E-10 | 55 | 8 |
| 7 | 8.25E-05 | 1.33E-10 | 31 | 7 |
| 8 | 1.18E-04 | 6.59E-09 | 38 | 19 |
| 9 | 1.64E-04 | 1.31E-10 | 56 | 6 |
| 10 | 1.10E-04 | 7.01E-09 | 35 | 21 |
| 11 | 1.37E-04 | 6.89E-09 | 43 | 20 |
| 12 | 1.45E-04 | 7.60E-09 | 47 | 22 |
| 13 | 1.30E-04 | 1.11E-10 | 40 | 5 |
| 14 | 1.50E-04 | 2.78E-09 | 50 | 10 |
| 15 | 1.15E-04 | 3.37E-09 | 37 | 13 |
| 16 | 1.90E-04 | 7.67E-09 | 60 | 23 |
| 17 | 1.43E-04 | 3.46E-09 | 46 | 14 |
| 18 | 1.60E-04 | 7.71E-09 | 53 | 24 |
| 19 | 1.59E-04 | 1.17E-08 | 52 | 30 |
| 20 | 1.42E-04 | 4.11E-09 | 45 | 15 |
| 21 | 1.48E-04 | 4.56E-09 | 49 | 16 |
| 22 | 1.35E-04 | 9.22E-09 | 41 | 26 |
| 23 | 1.23E-04 | 8.97E-11 | 39 | 4 |
| 24 | 1.83E-04 | 8.64E-11 | 59 | 3 |
| 25 | 1.71E-04 | 5.92E-09 | 57 | 18 |
| 26 | 1.46E-04 | 9.82E-09 | 48 | 28 |
| 27 | 9.57E-05 | 8.03E-11 | 33 | 2 |
| 28 | 9.01E-05 | 2.31E-09 | 32 | 9 |
| 29 | 1.02E-04 | 7.22E-11 | 34 | 1 |
| 30 | 1.36E-04 | 8.72E-09 | 42 | 25 |

**3. Adım**: Karar modeli

$\alpha$=0.05 ve çift yönlü test için Çizelge EK-1.3'e göre kritik değerler $0.5-\left(\alpha/2\right)$= *0.475*'ye karşılık gelen ±1.96 değeridir. Şekil 3.24'de karar modeli grafiksel olarak gösterilmektedir.




**Şekil 3.24.** Karar modeli

**4. Adım**: Karar

-6.65<-1.96 olduğu için hesaplanan *Z* değeri red bölgesine denk gelmektedir. Bundan dolayı $H_0$ hipotezi reddedilir ve $H_1$ hipotezi kabul edilir ($p$<0.05). Yani, CLPSO ve PCLPSO algoritmalarının farklı olduğuna 95% önem seviyesinde karar verilir. İstatistiksel yazılımlar hipoteze ait *p* değerlerini doğrudan verdiği için kullanımı tavsiye edilmektedir (Alpar, 2012). Bundan dolayı hipoteze ait *p* değerleri spss yazılımı kullanılarak hesaplanmıştır. Bu örnek için *p* değeri 0.000 olarak hesaplanmıştır.

### 3.5. Kruskal-Wallis Testi

Kruskal-Wallis testi, Wilcoxon sıra toplam testinin genişletilmiş şeklidir. İkiden fazla bağımsız örneğin farklı anakütlelerden gelip gelmediğine karar vermede kullanılır (Kartal, 2014). Testin adımları aşağıdaki şekildedir:

**1. Adım**: Hipotez kurulumu

$H_0$: incelenen özellik açısından anakütleler aynıdır.

$H_1$: incelenen özellik açısından anakütleler farklıdır.

**2. Adım**: Test istatistiği

Eğer *k* tane örnek var ise, bu örneklere ait tüm gözlem değerleri küçükten büyüğe sıralanır ve bunlara 1'den başlayarak ardışık olarak sıra numarası verilir. Her bir örneğe ait sıra numaraları toplanır ve bu toplamlar $T_j$ ile gösterilmek üzere test istatistiği Denklem (3.30) ile hesaplanmaktadır. Formüldeki $n_j$ ifadesi *j*. örnekteki gözlem adedini, *N* ifadesi ise toplam gözlem adedini temsil etmektedir.

$$H = \left[ \frac{12}{N(N+1)} \sum_{j=1}^{k} \frac{T_j^2}{n_j} \right] - 3(N+1) \tag{3.30}$$

**3. ve 4. Adım**: Karar modeli ve Karar

Hesaplanan $H$ değeri, Çizelge EK-2.1'deki ki-kare ($\chi^2$) tablosundan elde edilecek olan kritik değerle karşılaştırılır. Eğer $H > \chi^2_{\alpha\ ;\ k-1}$ ise $H_1$ hipotezi kabul edilir ($p<0.05$).

$H_0$ hipotezi reddedilirse, bu örneklerin alındıkları anakütlelerin farklı olduklarına $\alpha$ önem seviyesinde karar verilir. Hipoteze ait $p$ değeri spss yazılımı kullanılarak hesaplanmıştır.

# 4. ÖNERİLEN YÖNTEMLER

Bu bölümde, tez çalışmasında geliştirilmiş olan paralel metasezgisel algoritmalar ayrıntılı bir şekilde anlatılmaktadır. Bu bölümde ilk olarak, sürekli optimizasyon problemlerin çözümü için geliştirilen PCLPSO algoritması detaylı olarak anlatılmaktadır. PCLPSO algoritması çoklu-sürü, paralel çalışma ve işbirliği özelliklerine sahiptir. Çoklu-sürü tekniğinde, popülasyon alt popülasyonlara bölünmektedir ve her bir sürü birbirinden bağımsız olarak tüm arama uzayında çalışmaktadır. Sürüler en güncel çözümleri takas etmek için ve aramayı daha kaliteli çözümlere doğru yönlendirmek için periyodik olarak işbirliği yapmaktadır. Ayrıca, tez çalışmasında Açgözlü Bilgi Takası (ABT) diye adlandırılan yeni bir işbirliği stratejisi geliştirilmiştir ve bu bölümde anlatılmaktadır. PCLPSO algoritması, global optimizasyon üzerine çalışanlar tarafından iyi bilinen 14 tane kıyaslama fonksiyonu üzerinde test edilmiştir ve deneysel sonuçları Bölüm 5'de sunulmaktadır.

Bu bölümün devamında, ayrık optimizasyon problemi olan gezgin satıcı probleminin çözümü için geliştirilen PACO-3Opt algoritması ayrıntılı bir şekilde açıklanmaktadır. PACO-3Opt algoritması çoklu-koloni, paralellik ve işbirliği özelliklerine sahiptir. PACO-3Opt algoritması işbirliği stratejisi olarak ABT stratejisini kullanmaktadır. Bundan dolayı çözüm kalitesini artırma ve işlem süresini azaltma yeteneğine sahiptir. PACO-3Opt algoritması, hem küçük ölçekli hem de büyük ölçekli GSP problemleri üzerinde test edilmiştir ve deneysel sonuçları Bölüm 5'de sunulmaktadır.

## 4.1. Sürekli Optimizasyon Problemlerini Çözmek için Geliştirilen PSO Tabanlı Paralel Çoklu-Sürü Algoritması (PCLPSO)

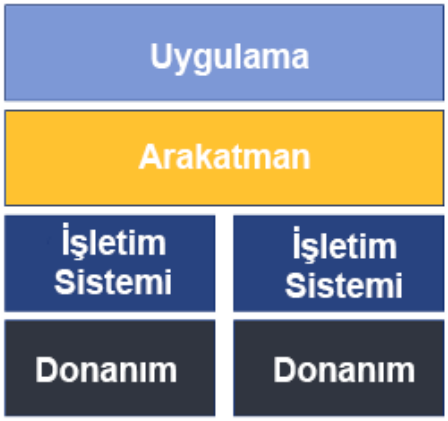
Son yıllarda, birçok araştırmacı özellikle bilgisayar bilimleri, yüksek başarımlı hesaplama, endüstri mühendisliği ve makine mühendisliği olmak üzere birçok alan için çok önemli olan optimizasyon konusu üzerinde çalışmaktadırlar. Optimizasyon problemleri genellikle NP-zor, karmaşık ve zaman alıcıdır (Torn ve Zilinskas, 1989; Talbi, 2009). Birçok metasezgisel algoritma optimizasyon problemlerini çözmek için geliştirilmiştir ve yeni algoritmalar da önerilmektedir. Sürü zekâsı ve evrimsel hesaplama, metasezgisel algoritmaların iki popüler alt sınıfıdır. Sürü zekâsı parçacık sürü optimizasyonu (Kennedy ve Eberhart, 1995), yapay karınca koloni algoritması

(Dorigo ve Stützle, 2004), yapay arı koloni algoritması (Karaboga ve Basturk, 2007), boz kurt optimizasyonu (Mirjalili ve ark., 2014) vb. yöntemleri içermektedir. Diğer taraftan, evrimsel hesaplama genetik algoritma (Goldberg, 1989), memetik algoritma (Neri ve ark., 2011) ve gen ifade algoritması (Ferreira, 2006) vb. yöntemleri içermektedir.

Bu çalışmanın Giriş bölümünde de bahsedildiği gibi, metasezgisel algoritmalar optimizasyon problemlerini çözmede çok etkilidir. Kennedy ve Eberhart (1995) tarafından geliştirilen PSO algoritması da bunlardan biridir. PSO popülasyon tabanlı ve metasezgisel bir optimizasyon tekniğidir. PSO algoritması kuş sürülerinin sosyal davranışlarından esinlenmiştir. Popülasyondaki her bir birey *parçacık* olarak adlandırılır ve potansiyel bir çözümü temsil etmektedir. Parçacıklar popülasyondaki mevcut en iyi çözümleri takip ederek arama uzayını taramaktadır ve böylece global optimuma yakınsamaktadır. PSO'nun başarısı ve popülerliği sayesinde PSO biyoinformatik (Yuan, 2015), enerji (Kıran ve ark., 2012), lojistik ve taşımacılık (Norouzi ve ark., 2017), işletme (Yang ve ark., 2011), kimya mühendisliği (Schwaab ve ark., 2008), otomotiv endüstrisi (Yildiz, 2012) gibi birçok farklı alanda kullanılmaktadır.

Bilim ve teknolojideki hızlı ilerlemeler bütün alanlardaki gibi hesaplama alanındaki gelişmeleri de tetiklemektedir. Bu gelişmelerin sonucu olarak, bilgisayarlar tarafından çözülmesi gereken problemler daha büyük ve daha karmaşık olmaktadır ve bunlar büyük ölçekli verilerin artmasına neden olmaktadır. Bundan dolayı, bu problemleri çözmek ve büyük ölçekli verileri işlemek için paralel hesaplamaya ve paralel algoritmalara ihtiyaç duyulmaktadır.

Paralel hesaplamada, alt görevlere ayrılmış bir görev sonuçları daha çabuk elde etmek için çoklu işlemcilerde eşzamanlı olarak çalıştırılmaktadır (Grama, 2003). Paralel hesaplama kullanılarak algoritmaların performansı artmaktadır ve büyük ölçekli problemler daha kısa zamanda çözülmektedir. Bundan dolayı, paralel hesaplama veriler günden güne artmakta olduğu için günümüzde tıbbi görüntü işleme (Romero-Laorden ve ark., 2016), finans (Balakrishnan ve ark., 2014), biyoinformatik (Orobitg ve ark., 2015) gibi farklı birçok alanda kullanılmaktadır. Bu çalışmada, önerilen paralel algoritmayı geliştirmek için bir arakatman (middleware) olan Jade yazılım çatısı (Bellifemine ve ark., 2007) kullanılmaktadır. Arakatman; farklı bilgisayarları, ağları ve işletim sistemleri destekleyen ve paralel uygulamalar geliştirmeyi mümkün kılan Şekil 4.1'de görüldüğü gibi işletim sistemleri ve uygulamalar arasındaki bir yazılım katmanıdır (Tanenbaum ve Van Steen, 2007).




**Şekil 4.1.** Arakatman

Metasezgisel optimizasyon algoritmaları genellikle sıralı bir yapıya sahiptir. Onların kullanımı zaman karmaşıklığını düşürmesine rağmen, havacılıktaki (Olhofer ve ark., 2001; Hasenjäger ve ark., 2005) gibi akademide ve endüstride meydana çıkan bazı gerçek dünya problemlerinin çözümleri hala çok uzun zaman almaktadır. Bu yüzden, paralel hesaplama hem arama süresini azaltmak hem de çözümlerin kalitesini artırmak için metasezgisel algoritmalar ile beraber kullanılmaktadır (Alba, 2005). Bu çalışmanın amacı, global optimizasyonun alt sınıfı olan kısıtsız global optimizasyon problemlerini (Schäffler, 2012) özellikle büyük ölçekli problemleri çözmek için yeni bir paralel metasezgisel algoritma geliştirmektir.

Büyük ölçekli optimizasyon problemleri için çözüm bulma süreci aşağıdaki önemli sorulara yol açmaktadır:

(i) Makul bir süre içerisinde problemleri çözmek için ne tür paralel metasezgisel algoritmalar kullanılmalıdır?

(ii) Klasik optimizasyon yöntemleri kullanarak elde edilen sonuçlardan daha iyi sonuçlar paralel metasezgisel algoritma kullanılarak nasıl elde edilebilir?

(iii) Paralel metasezgisel algoritma oluşturmak için ihtiyaç duyulan işbirliği, topoloji, göç seçme ve etkileşim stratejileri nasıl olmalıdır ve bunların parametrelerinin en uygun değerleri nelerdir?

(iv) Bu yöntem lokal optimumlara takılmayı engelleyecek midir?

Literatür taramasından sonra, CLPSO algoritmasının (Liang ve ark., 2006) diğer PSO türevlerinden daha iyi performansa sahip olduğu görülmüştür. Daha önce bahsedildiği gibi, arama uzayının boyutu arttıkça birçok optimizasyon algoritmasının etkisi ve verimliliği kötüleşmektedir. Metasezgisel algoritmaların kullanımı zaman

karmaşıklığını düşürmesine rağmen hala birçok gerçek dünya problemlerinin çözümleri zor olmaktadır ve çok uzun zaman almaktadır (Dolan ve ark., 2004). Bundan dolayı bu zorlukların üstesinden gelmek için, paralel hesaplama kullanılarak CLPSO algoritmasının performansının iyileştirilmesi amaçlanmıştır. Bu çalışmada çözüm kalitesini ve kararlılığı artıran, arama süresini azaltan yeni paralel çoklu-sürü CLPSO algoritması (PCLPSO) önerilmektedir. PCLPSO algoritmasının başarısı kıyaslama fonksiyonları üzerinde gösterilmiştir. Bu çalışmanın literatüre temel katkıları şunlardır:

(i) Geliştirilen algoritma yeni bir işbirliği stratejisi kullanmaktadır.

(ii) Geliştirilen algoritma aramayı önemli derecede hızlandırmaktadır.

(iii) Geliştirilen algoritma elde edilen çözümlerin kalitesini artırmaktadır.

(iv) Geliştirilen algoritma kararlılığı artırmaktadır.

(v) Geliştirilen algoritma ayrıca büyük ölçekli problemleri de başarılı şekilde çözmektedir.

PCLPSO algoritması çoklu-sürü ve işbirliği özellikleri sayesinde çözüm kalitesini ve global arama kabiliyetini artırmaktadır. Ayrıca, dağıtık sistem üzerinde paralel çalışması sayesinde hesaplama süresini düşürmektedir. PCLPSO algoritması aşağıdaki alt bölümlerde detaylı olarak anlatılmıştır.

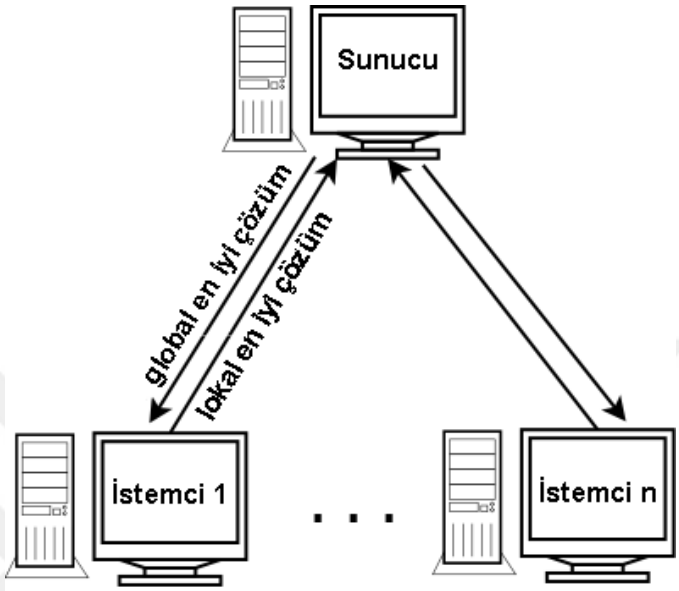
### 4.1.1. Çoklu-sürü stratejisi

Çoklu-sürü tekniğinde, bir popülasyon alt popülasyonlara bölünmektedir. Her bir alt popülasyon bir sürüyü temsil etmektedir. Algoritmada iki tip sürü bulunmaktadır: sunucu-sürü ve istemci-sürü. Her bir sürü $s_i$, $i \in (1, 2,\ldots, n)$, bağımsız olarak PCLPSO algoritmasını çalıştırmaktadır. Böylece, her bir sürü $s_i$ birbirinden bağımsız olarak tüm arama uzayında çalışmaktadır.

### 4.1.2. Paralellik stratejisi

PCLPSO'nun paralellik özelliği; her bir $s_i$ sürünün algoritmayı aynı zamanda farklı bilgisayarlarda çalıştırmasıdır. Böylece, hesaplama süresi düşmektedir. Şekil 4.2'de dağıtık sistemin küme yapısı gösterilmektedir. PCLPSO algoritması sadece bir tane sunucu-sürü ve birkaç tane istemci-sürü içermektedir. İstemci-sürü sayısı problemlerin zorluğuna göre değişmektedir. Sunucu-sürü dâhil her bir sürü global optimuma ulaşmak için PCLPSO algoritmasını işbirliğiyle ve eşzamanlı olarak

çalıştırmaktadır. Paralelliği yerine getirmek için bir arakatman olan Jade yazılım çatısı (Bellifemine ve ark., 2007) kullanılmaktadır. Ayrıca Jade, hata ayıklama ve uygulamayı yükleme aşamalarını destekleyen araçları sayesinde dağıtık sistemlerin gerçekleştirilmesini kolaylaştırmaktadır.

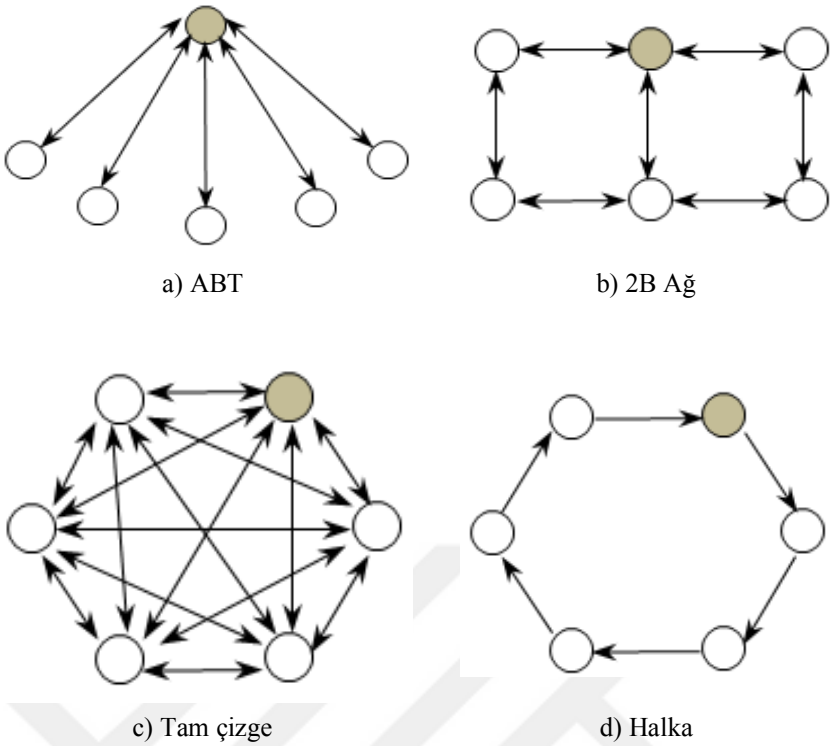



**Şekil 4.2.** Küme yapısı

### 4.1.3. İşbirliği stratejisi

İşbirliği tekniğinde, PCLPSO algoritmasındaki *n* adet sürü, en güncel çözümleri takas etmek için ve aramayı daha kaliteli çözümlere doğru yönlendirmek için periyodik olarak işbirliği yapmaktadır. Literatürde birkaç tane önerilmiş işbirliği stratejisi bulunmaktadır. Bunlardan birkaçı halka, 2B ağ ve tam çizgedir (Talbi, 2009) ve bunların topolojileri Şekil 4.3'de gösterilmektedir.

Bu tez çalışmasında paralel metasezgisel PCLPSO algoritması geliştirilirken Açgözlü Bilgi Takası (ABT) diye adlandırılan yeni bir işbirliği stratejisi geliştirilmiştir ve literatüre kazandırılmıştır. ABT'nin başarısını etkileyen dört faktör bulunmaktadır: göç periyodu, topoloji, takas edilecek bilgi ve yer değiştirme poliçesi. Bu süreç Çizelge 4.1'deki algoritmanın sözde kodunda süslü parantez ile vurgulanmaktadır (satır 27-33). Ayrıca ABT ile halka, 2B ağ ve tam çizge işbirliği stratejilerinin topolojileri karşılaştırılmıştır ve elde edilen sonuçlar Deneysel Sonuçlar bölümünde Çizelge 5.14 ve Şekil 5.5'de verilmiştir. ABT'nin topolojisinin diğer işbirliği stratejilerinin topolojilerinden daha başarılı olduğu deney sonuçlarından görülmektedir.



a) ABT b) 2B Ağ

c) Tam çizge d) Halka

**Şekil 4.3.** Göç topolojisi (Dolu daire sunucu sürüyü temsil etmektedir.)

ABT'nin başarısını etkileyen birinci faktör göç periyodudur. Göç sürecinde çözüm kalitesi ile hesaplama süresi arasında daima bir ödün verme söz konusudur. Eğer bilgi çok sık takas yapılıyorsa, o zaman PCLPSO algoritmasının çözüm kalitesi daha iyi olabilmektedir. Fakat PCLPSO algoritmasının hesaplama süresi artmaktadır. Eğer göç aralığı daha genişse algoritmanın hesaplama süresi düşmektedir. Fakat algoritmanın çözüm kalitesi kötüleşebilmektedir. Göç periyodunu belirlemenin iki çeşit stratejisi vardır: *Kör* ve *adaptif* (Talbi, 2009). Adaptif stratejide başlangıç aşamalarında sürüler arasında bilgi alışverişi bulunmakta, fakat sonlara doğru sürüler arasındaki bilgi alışverişi kesilmektedir. Bundan dolayı bu çalışmada *kör stratejisi* tercih edilmektedir. Bu stratejide göç süreci periyodik olarak belirli bir iterasyon sayısı sonrasında oluşmaktadır (Algoritma 1, satır 27). Uygun bir $P$ göç periyodu belirlemek için, 10-B problemler üzerinde farklı iterasyon aralıkları kullanılarak bir deney gerçekleştirilmiştir ve sonuçlar Çizelge 5.4'de gösterilmektedir.

İkinci faktör bilgileri paylaşmak için komşu sürülerin belirlendiği topolojidir. Eğer bilgiler çok fazla sürü arasında paylaşılırsa algoritmanın hesaplama süresi doğal olarak artmaktadır. Eğer bilgiler çok az sürü arasında paylaşılırsa hesaplama süresi azalmaktadır, fakat lokal optimumlara takılı kalma olasılığı artabilmektedir. ABT

işbirliği stratejisinin topolojisi Şekil 4.3'de gösterilmektedir. Şekil 4.3'den de görüldüğü gibi bu topolojide haberleşmeler sadece sunucu-sürü ile istemci sürüler arasındadır (Algoritma 1, satır 8, 28 ve 31) ve istemci sürüler arasında doğrudan bir haberleşme bulunmamaktadır. Bu topolojide düşük haberleşme frekansı olduğu için yüksek hesaplama verimliliği elde edilmektedir. ABT, halka, 2B ağ ve tam çizge işbirliği stratejilerinin topolojileri karşılaştırılmıştır ve sonuçlar Çizelge 5.14 ve Şekil 5.5'de verilmiştir. Hem hesaplama süresi hem de çözümlerin kalitesi açısından, ABT'nin topolojisinin diğer işbirliği stratejilerinin topolojilerinden daha başarılı olduğu deney sonuçlarından görülmektedir.

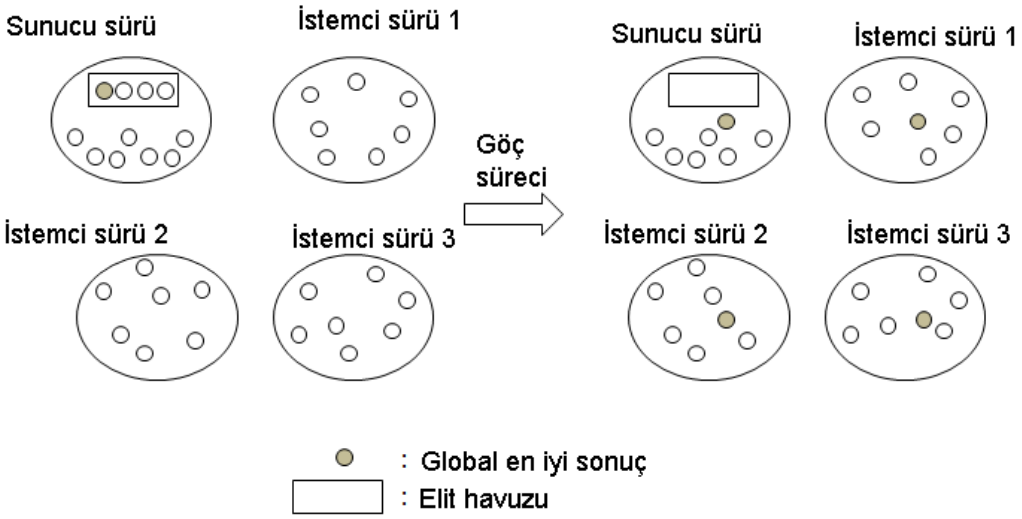
**Çizelge 4.1.** PCLPSO Algoritmasının sözde kodu

**Algoritma 1:** PCLPSO Algoritması

```
1:  int n                                   /* sürü sayısı */
2:  int k                                   /* her bir sürünün popülasyon boyutu */
3:  int P                                   /* göç periyodu */
4:  Array EH                                /* ElitHavuzu, lokal en iyi çözümleri tutar */
5:  int m                                   /* boşluk tazeleme */
6:  double w, c1, c2                        /* eylemsizlik ağırlığı ve ivmelenme katsayıları */
7:  n, k, P, m, w, c1, c2 Parametrelerine ilk değer ata
8:  Sunucu Parametreleri istemcilere gönderir
9:  parfor i ← 1, n do
10:     i. sürünün parametrelerine ilk değer ata
11:     for j ← 1, k do
12:         Konum ve hıza ilk değer ata
13:         Uygunluğu hesapla
14:     end for
15: end parfor
16: int d=0;                                /* İterasyon sayacına ilk değer ata*/
17: repeat:
18:     d++;                                /* İterasyon sayacını artır */
19:     parfor i ← 1, n do
20:         for j ← 1, k do
21:             Denklem (3.1)'i kullanarak hızı güncelle
22:             Denklem (3.2)'i kullanarak konumu güncelle
23:             Uygunluğu hesapla
24:         end for
25:         lBest_i'i güncelle   /* i. sürünün lBest'ini güncelle */
26:     end parfor
27:     if d mod P == 0 then
28:         Her bir istemci sürü lBest'i sunucu sürüye gönderir
29:         Sunucu lBest'leri EH'ye kaydeder
30:         Sunucu EH'deki gBest'i bulur ve EH'yi boşaltır          } İşbirliği
31:         Sunucu gBest'i istemcilere gönderir
32:         Her bir sürü rasgele seçilen bir parçacığı gBest ile değiştirir
33:     end if
34: until durdurma kriteri sağlandı
35: return gBest
```

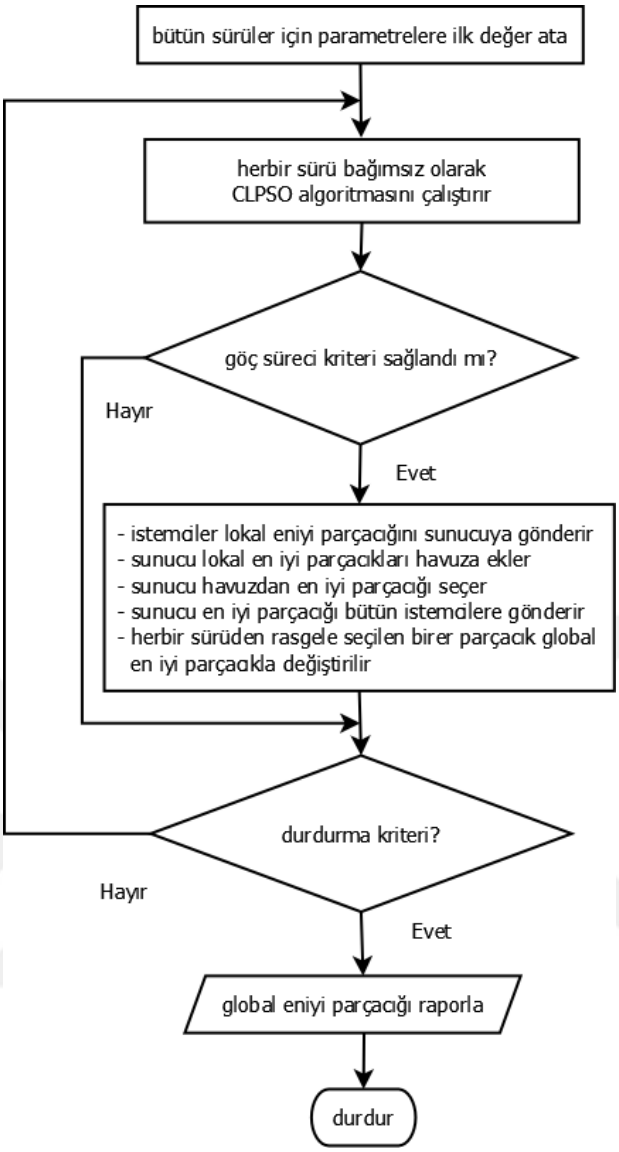
Üçüncü faktör algoritmanın performansını önemli ölçüde etkileyen takas edilecek bilgidir. Genellikle, sürüler arasında değiştirilecek bilgiler global en iyi çözüm, lokal en iyi çözümler veya komşuluk en iyi çözümü olabilmektedir. ABT stratejisinde, her bir sürü kendi lokal en iyi çözümünü (*lBest*) PCLPSO'nun göç sürecinde sunucu sürüye göndermektedir (satır 28). Sunucu kendi *lBest* çözümü dâhil kendisine gelen bütün *lBest* çözümleri *elit havuzu* (EH) diye adlandırılan bir havuzun içerisinde toplamaktadır (satır 29). Daha sonra sunucu sürü EH içinden en iyi çözümü seçmektedir (satır 30). Popülasyondaki global en iyi çözüm (*gBest*) olan bu çözüm sunucu sürü tarafından bütün istemci sürülere gönderilmektedir (satır 31). Şekil 4.4 bu göç sürecini tasvir etmektedir. Tek-sürüye sahip PSO tabanlı bir algoritmada, eğer *gBest* arka arkaya birkaç iterasyonda iyileştirilemezse bu algoritmanın durağan duruma düşme olasılığı her zaman mevcuttur. Fakat çoklu-sürüye sahip PCLPSO algoritmasında aramayı daha kaliteli çözümlere doğru yönlendirmek için *gBest* çözümü sürüler arasında paylaşılmaktadır. Göç sürecinden sonra, her bir sürü $s_i$, daha iyi *lBest* çözümü kullanarak global optimumu aramaya devam etmektedir. Eğer PCLPSO'da herhangi bir sürüdeki *lBest* çözümü lokal optimuma takılırsa, bu strateji sayesinde lokal optimuma takılan bu sürü lokal optimumdan rahatça kurtulmaktadır. Bundan dolayı, PCLPSO daha iyi ve daha kararlı çözümler elde etmektedir.

Dördüncü faktör alınan bilgilerin nasıl kullanılacağıyla ilgilenen yer değiştirme poliçesidir. Bu aşamada, her bir istemci sürüden rasgele seçilen birer parçacık sunucu tarafından gönderilmiş olan *gBest* ile değiştirilmektedir. Sunucu sürüde rasgele seçilen bir parçacık da *gBest* ile değiştirilmektedir (satır 32). Böylece, Şekil 4.4'de gösterildiği gibi PCLPSO çözüm kalitesini geliştirmektedir.

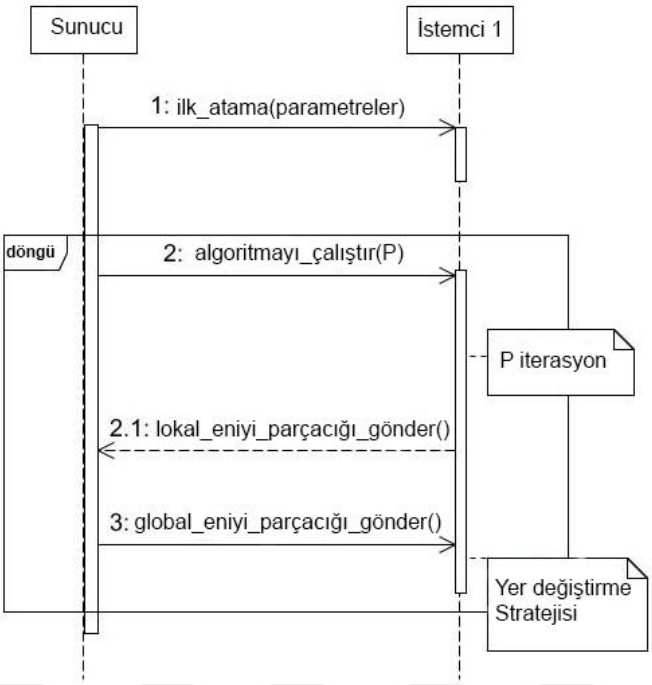



**Şekil 4.4.** Göç süreci

PCLPSO algoritmasının sözde kodu, akış diyagramı ve sıra diyagramı sırasıyla Çizelge 4.1, Şekil 4.5 ve Şekil 4.6'da gösterilmektedir. Sıra diyagramındaki senaryo, algoritmanın gerçekleştirilmesini ve sunucu ile istemci arasındaki haberleşmeyi göstermektedir. Öncelikle bilgisayarların sayısı yani sürü sayısı $n$, problemin zorluğuna göre belirlenmektedir (Algoritma 1, satır 1). Bunlardan bir tanesi sunucu sürü içindir ve diğerleri istemci sürüleri içindir. Daha sonra, algoritmanın parametreleri belirlenmektedir ve ilk değerleri atanmaktadır (satır 2-7): $P$ göç periyodu, maksimum iterasyon sayısı, popülasyon boyutu, eylemsizlik ağırlığı, $m$ boşluk tazeleme, $c_1$ ve $c_2$ ivmelenme katsayıları. Sunucu bu parametreleri istemcilerle algoritmanın başında paylaşmaktadır (satır 8). Her bir sürü $s_i$ başlangıç parçacıklarını üretmektedir ve bu parçacıkların uygunluk değerlerini hesaplamaktadır (satır 9-15). İlk değer atama işleminden sonra haberleşme döngüsü başlamaktadır (satır 17). Sunucu da dâhil her bir sürü algoritmayı $P$ iterasyona ulaşıncaya kadar çalıştırır (satır 19-26). $P$. iterasyonda göç süreci meydana gelir. Göç sürecinde; istemciler kendi lokal en iyi çözümlerini sunucuya göndermektedirler, sunucu bunları bir havuzda toplamaktadır, daha sonra bunların arasındaki global en iyi çözümü bulmaktadır ve onu istemcilerle paylaşmaktadır (satır 27-33). Bu haberleşme döngüsü, algoritma maksimum iterasyon sayısına ulaşıncaya kadar devam etmektedir (satır 34). Algoritmanın sonunda, global en iyi parçacık çözüm olarak kullanılmaktadır (satır 35).




**Şekil 4.5.** PCLPSO algoritmasının akış diyagramı




**Şekil 4.6.** PCLPSO algoritmasının sıra diyagramı

## 4.2. Gezgin Satıcı Probleminin Çözümü için ACO ve 3-Opt Tabanlı Paralel İşbirlikçi Bir Yöntem

Hesaplamalı karmaşıklık teorisi bilgisayar bilimi ve matematikte temel bir daldır. Hesaplamalı karmaşıklık teorisi, hesaplamalı problemlerin zorluk derecesine göre hesaplamalı problemleri sınıflandırmaya odaklanmaktadır. Bu problemlerden bir tanesi de kombinasyonlu optimizasyon problemi olan GSP'dir. GSP'de, bir gezgin satıcı bütün şehirlere sadece bir kez uğramayı ve başladığı şehire geri dönerek kapalı bir tur oluşturmayı amaçlamaktadır. Bu tur, gezilen yolun uzunluğunu temsil etmektedir. Şehir sayısı arttıkça optimum tur uzunluğunu belirlemek çok zorlaşmaktadır. Bu nedenle bir GSP karar problemi genellikle NP-tam problemi olarak sınıflandırılmaktadır (Donald, 2011). Metasezgisel algoritmalar genellikle bir çözüm havuzu üzerinde çalışır ve en uygun çözümü elde etmeyi amaçlamaktadır. Bu nedenle, çözülen bu problemler optimizasyon problemleri olarak sınıflandırılır. GSP, bir optimizasyon problemi olarak çözüldüğünde NP-zor problem olarak sınıflandırılmaktadır (Donald, 2011).

GSP tam sonucu veren algoritmalar veya sezgisel yöntemler kullanılarak çözülebilmektedir. Tam sonucu veren algoritmalar büyük ölçekli GSP için uygun değildir. Birkaç sezgisel algoritma ise büyük ölçekli GSP'nin çözümü için geliştirilmiştir (Dorigo ve Gambardella, 1997; Marinakis ve ark., 2011; Feng ve ark.,

2013; Avşar ve Aliabadi, 2015). Son yıllarda, GSP'nin daha iyi çözümlerini elde etmek için paralel ve hibrit algoritmalar da önerilmiştir. Bunlardan birkaçı Kaynak Araştırması bölümünde listelenmektedir. Sezgisel yaklaşımlar ve hibrit yöntemler GSP'nin çözümünde iyi sonuçlar elde etmelerine rağmen, bunlar lokal optimumlardan başarılı şekilde kaçamamaktadırlar. Ayrıca, tek koloniye sahip ACO algoritmasının en büyük eksikliği aramanın erken durgunluğudur (Dorigo ve ark., 1996; Kolli ve Sharvani, 2014). Bunun yanında, onların çalışma süreleri ne yazık ki çok uzun zaman almaktadır. Bu eksikliklerin üstesinden gelmek ve çözüm kalitesini artırmak için 3-Opt algoritmasını kullanan karınca koloni optimizasyonuna dayalı çoklu-koloniye sahip ve paralel çalışan paralel işbirlikçi hibrit bir algoritma (PACO-3Opt) geliştirilmiştir (Gülcü ve ark., 2016). Bu çalışmanın temel katkıları şunlardır: Bu yeni yöntem çözüm kalitesini artırma ve işlem süresini azaltma yeteneğine sahiptir. PACO-3Opt algoritması onun çoklu-koloni ve işbirliği özelliği sayesinde çözümlerin kalitesini ve kararlılığı artırmaktadır. Ayrıca, PACO-3Opt algoritması dağıtık hesaplama ortamında paralel olarak çalışma yeteneğiyle hesaplama süresi düşmektedir.

Geliştirilen algoritmanın çoklu-koloni özelliğinde, popülasyon alt popülasyonlara bölünmektedir. Her bir alt popülasyon bir karınca kolonisini temsil etmektedir ve bu koloninin tipi ya sunucu koloni ya da istemci koloni olabilmektedir. Sadece bir tane sunucu koloni vardır ve diğer koloniler istemcidir. Toplam koloni sayısı problemlerin zorluk derecesine göre belirlenmelidir. Yukarıda da bahsedildiği gibi, tek koloniye sahip ACO algoritmasının en büyük eksikliği aramanın erken durgunluğudur (Dorigo ve ark., 1996; Kolli ve Sharvani, 2014). Durgunluk; bütün karıncaların aynı rotayı takip etmesi ve aynı çözümü tekrar tekrar oluşturmasındaki durum olarak tarif edilmektedir (Dorigo ve ark., 1996). Bu eksikliğin üstesinden gelmek için, çoklu-koloniye sahip PACO-3Opt algoritmasında global en iyi çözüm koloniler arasında ara ara paylaşılmaktadır. Böylece, algoritma aramayı daha kaliteli çözümlere doğru yönlendirmektedir. Eğer bir koloni durgun bir durumda ise diğer koloniler bu koloniyi durgun durumdan kurtarmak için yardım etmektedirler. Böylece, PACO-3Opt algoritması daha iyi ve daha kararlı sonuçlar elde etmektedir.

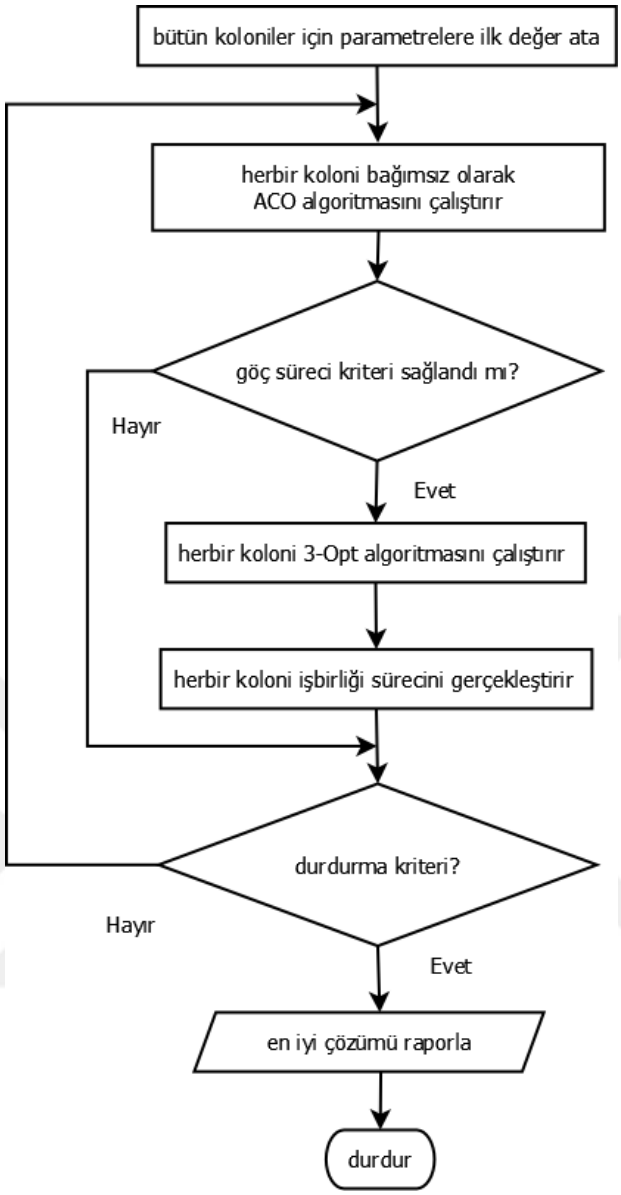
Algoritmanın paralelliği ise her bir koloni optimal turu bulmak için PACO-3Opt algoritmasını aynı zamanda farklı bilgisayarlarda bağımsız olarak çalıştırmasıdır. Paralelliği gerçekleştirmek için Jade yazılım çatısı (Bellifemine ve ark., 2007) kullanılmaktadır. İşbirliği stratejisinde, bütün koloniler daha iyi ve daha kararlı

çözümler üretmek için bulunan en güncel çözümleri periyodik olarak takas yapmaktadırlar.

Literatürde topolojileri Şekil 4.3'de gösterilen Açgözlü Bilgi Takası (ABT) (Gülcü ve Kodaz, 2015), halka, 2B ağ ve tam çizge gibi birkaç işbirliği stratejisi bulunmaktadır. Hatta birkaç çalışma işbirliği stratejilerini çözüm kalitesi ve hesaplama zamanı özellikleri açısından karşılaştırmaktadır (Tang ve ark., 2004; Ruciński ve ark., 2010; Nalepa ve ark., 2013). ABT stratejisinde, istemci koloniler arasında doğrudan bir iletişim yoktur. Bundan dolayı, ABT stratejisinin haberleşme frekansı 2B ağ ve tam çizgeden daha düşüktür. Bunun yanında, ABT stratejisindeki her bir koloni komşu koloniler tarafından paylaşılan bilgiye aynı anda sahip olmaktadır ve bu bilgiyi kullanmaktadır. Halka ve 2B ağ bu özelliğe sahip değildir. ABT stratejisinin bu avantajlarından dolayı, PACO-3Opt algoritması işbirliği stratejisi olarak tez çalışmasında geliştirilen ABT stratejisini kullanmaktadır. Algoritmanın performansını önemli ölçüde etkileyen ABT stratejisinin elemanları göç periyodu, göç topolojisi, takas edilecek bilgi ve yer değiştirme politikasıdır. İşbirliği süreci Çizelge 4.2'de sunulan algoritmanın satır 19-23 satırlarında gösterilmektedir. Şekil 4.3'deki ABT topolojisinden görüldüğü gibi, bilgi takası sadece sunucu koloni ve istemci koloniler arasındadır (Algoritma 2, satır 2, 19 ve 22). Göç periyodu *P* göç sürecinin hangi sıklıkla oluşacağını belirtmektedir (Algoritma 2, satır 16). Dağıtık hafıza mimarisi kullanıldığından dolayı ağ üzerinden bilgiyi gönderme ve alma arasında geçen bir zaman aralığı söz konusudur. Bundan dolayı, kolonilerin haberleşme sıklığı hesaplama zamanını ve çözüm kalitesini etkilemektedir ve her zaman bunlar arasında bir ödün verme söz konusudur. Eğer göç periyodu kısa ise yani göç çok sık gerçekleşiyorsa, koloniler bilgiyi çok sık takas edeceklerdir. Dolayısıyla, çözüm kalitesi daha iyi olabilir, fakat hesaplama süresi doğal olarak artmaktadır. Diğer taraftan, eğer göç periyodu daha uzun ise hesaplama süresi düşmektedir, fakat çözüm kalitesi daha kötü olabilmektedir. PACO-3Opt algoritmasında en uygun göç periyodunu belirlemek için bazı deneyler gerçekleştirilmiştir ve bu deneylerin sonuçları Çizelge 5.24 ve Şekil 5.6'da gösterilmektedir. Sunucu koloni ve istemci koloniler algoritmanın çalışması boyunca *lbestTour* ve *gbestTour* değerlerini takas etmektedirler. *lbestTour* bir kolonideki en iyi turdur ve *gbestTour* bütün koloniler arasındaki en iyi turdur. Takas işleminden sonra, her bir koloni *gbestTour*'a ait feromon değerini turdaki yollara bırakmaktadır.

Çizelge 4.2'de PACO-3Opt algoritmasının sözde kodu verilmiştir. Bilgisayar sayısını temsil eden koloni sayısı *n*, her bir kolonin popülasyon boyutu *m* ve göç

periyodu *P* kullanıcı tarafından belirlenmektedir ve algoritmaya giriş parametresi olarak verilmektedir. Algoritmanın başında, ACO algoritmasının parametrelerine ilk değerleri atanır (satır 1). Bu parametreler $\alpha$, $\beta$, feromon buharlaşma katsayısı $\rho$ ve $Q$ sabitidir. Denklem (3.6)'da görüldüğü gibi $\alpha$ ve $\beta$ parametreleri iki şehir arasındaki uzaklığın ve feromon izinin önem derecesini belirlemektedir. Daha sonra, sunucu koloni bu parametreleri istemcilere göndermektedir (satır 2). Bütün şehirler arasındaki feromon miktarı belirlenmektedir (satır 8) ve karıncaların hepsi farklı şehirlere rasgele dağıtılmaktadır (satır 9). Her bir karınca kendi turunu tamamlamaktadır (satır 12). Bütün karıncalar turlarını tamamladıktan sonra her bir karınca geçtiği bütün yollara bırakacağı feromonu hesaplar (satır 14). Daha sonra, bütün karıncalar tarafından şehirler arasında bırakılacak toplam feromon miktarı hesaplanır, buharlaşma süreci gerçekleşir ve bütün şehirler arasındaki feromon miktarı güncellenir (satır 15). Böylece, her bir koloni bir iterasyonu tamamladığı zaman iterasyon sayacı artırılır (satır 27) ve sonraki iterasyon başlar. Göç süreci her *P.* iterasyonda bir tekrarlamaktadır (satır 16). Göç sürecinin başında, her bir koloni ACO tarafından bulunan turları geliştirmek için 3-Opt algoritmasını çalıştırmaktadır (satır 17) ve *lbestTour*'u bulmaktadır (satır 18). Her bir koloni kendi *lbestTour*'unu sunucuya göndermektedir (satır 19). Sunucu *lbestTour*ları elit havuzunda (EH) depolar (satır 20). Sunucu EH'deki *gbestTour*'u bulur ve EH'yi boşaltır (satır 21) ve *gbestTour*'u istemci kolonilere gönderir (satır 22). Her bir koloni *gbestTour* ile rasgele seçilen bir turu yer değiştirir (satır 23). Yer değiştirme işleminden sonra, her bir koloni *gbestTour*'a ait feromon değerini turdaki yollara bırakır. Böylece, lokal optimuma düşmüş bir koloni işbirliği sayesinde bu lokal optimumdan kolayca kurtulmaktadır. Bu süreç durdurma kriteri sağlanıncaya kadar devam eder (satır 28). Algoritmanın sonunda, en iyi tur çözüm olarak geri döndürülür (satır 29). Şekil 4.7 geliştirilen algoritmanın akış diyagramını göstermektedir.




**Şekil 4.7.** PACO-3Opt yönteminin akış diyagramı

ACO algoritmasının davranışı verilen parametrelerin değerlerine bağlıdır (Dorigo ve Stützle, 2004). Birçok ACO uygulamasında, parametre değerleri algoritmanın her bir koşması boyunca değişmeden aynı kalmaktadır. $\alpha$ ve $\beta$ parametreleri iki şehir arasındaki uzaklığın ve feromon izinin önem derecesini belirlemektedir. $\alpha$ parametresi ne kadar büyükse sezgisel bilgi de o kadar büyüktür. Fakat $\alpha$ parametresi aşırı derecede küçük kullanıldığı zaman karıncalar lokal en iyiye yönlendirileceklerdir. $\beta$ parametresi ACO'nun yakınsama özelliği üzerinde büyük etkiye sahiptir. $\beta$ parametresi aşırı derecede küçük olduğu zaman iterasyon sayısı artacaktır ve yakınsama hızı yavaşlayacaktır. Bu durumda, optimal çözümü bulmak zordur. Diğer taraftan, $\beta$ parametresi aşırı derecede büyük olduğu zaman ACO'nun yakınsama özelliği

kötüleşme eğilimindedir. Aslında, $\beta$ parametresi $\alpha$ parametresi ile güçlü bir ilişkiye sahiptir (Duan ve ark., 2007). ACO'nun $\alpha$ ve $\beta$ parametre değerleri hakkında literatürde (Hao ve ark., 2006; Duan ve ark., 2007; Stützle ve ark., 2011; Peker ve ark., 2013; Mahi ve ark., 2015) gibi çalışmalar bulunmaktadır. Mahi ve ark. (2015) küçük-ölçekli problemler için $\alpha$ ve $\beta$ parametrelerinin en iyi değerlerini önermişlerdir. Bu çalışmada, küçük-ölçekli GSP örnekleri için Mahi ve ark. tarafından önerilen değerler kullanılmıştır ve bu değerler Çizelge 5.23'de gösterilmektedir. Ayrıca, Yong (2015) büyük-ölçekli GSP örnekleri için $\alpha$ ve $\beta$ parametre değerlerinin sırasıyla 3 ve 2'ye eşit olduğu zaman daha iyi sonuçların elde edildiğini belirtmektedir. Bundan dolayı deneylerde büyük-ölçekli GSP örnekleri için $\alpha$ ve $\beta$ parametre değerleri sırasıyla 3 ve 2'ye eşitlenmiştir.

**Çizelge 4.2.** PACO-3Opt algoritmasının sözde kodu

**Algoritma 2:** PACO-3Opt Algoritması

**Giriş:** *n*: koloni sayısı, *m*: her bir kolonin popülasyon boyutu, *P*: göç periyodu
**Çıkış:** global en iyi çözüm *gbestTour*

```
ACO'nun α, β, p, Q parametrelerine ilk değer ata
Sunucu Parametreleri istemcilere gönderir
d ← 0;                                          // iterasyon sayacı
repeat
    parfor i ← 1, n do
        if d == 0 then
            i. kolonin parametrelerine ilk değer ata
            Feromon izine ilk değer ata
            i. kolonin bütün karıncalarını rasgele dağıt
        end if
        for k ← 1, m do
            k. karınca Denklem (3.6)'i kullanarak bir tur tamamlar
        end for
        Her bir karınca Denklem (3.7)'i kullanarak bırakacağı feromonu hesaplar
        Denklem (3.8) ve (3.9)'i kullanarak yoldaki feromonu günceller
        if d mod P == 0 then
            3-Opt algoritmasını çalıştır        // turları iyileştir
            lbestTour_i bul                     // i. kolonin lokal en iyi turunu bul
            Her bir koloni kendi lbestTour değerini sunucuya gönderir
            Sunucu lbestTour değerlerini EH'ye kaydeder
            Sunucu EH'deki gbestTour değerini bulur ve EH'yi boşaltır
            Sunucu gbestTour değerini istemcilere gönderir
            Her bir koloni rasgele seçilmiş bir turu gbestTour ile değiştirir
            Her bir koloni gbestTour'a ait feromonu Denklem (3.7)'ye göre bırakır
        end if
    end parfor
    d ← d + 1;
until durdurma kriteri sağlandı
return gbestTour
```

# 5. DENEYSEL SONUÇLAR VE TARTIŞMA

Bu bölümde tez çalışmasında geliştirilmiş olan PCLPSO ve PACO-3Opt algoritmalarının ve ABT işbirliği stratejisinin deneysel sonuçları sunulmaktadır ve elde edilen sonuçların yorumları verilmektedir.

## 5.1. PCLPSO Algoritmasının Deneysel Sonuçları

Bu bölümde, PCLPSO algoritmasının deneysel sonuçları sunulmaktadır. PCLPSO algoritmasının performansı 14 adet kıyaslama fonksiyonu üzerinde test edilmiştir. Formülleri ve özellikleri Materyal ve Metot bölümünde verilen bu fonksiyonlar CLPSO'da kullanıldığından dolayı seçildi. Bu fonksiyonlar global optimizasyon üzerine çalışanlar tarafından iyi bilinmektedirler ve optimizasyon algoritmalarının testi ve analizi için sıklıkla kullanılmaktadırlar. Bu 14 fonksiyonun global optimum değerleri, ilk değer alma aralığı ve arama aralığı Çizelge 3.1'de verilmektedir. Deneylerde her bir test fonksiyonun boyutu 10, 30 ve 100 olarak ayarlanmıştır. Çizelge 5.1 algoritmanın geliştirilmesinde ve deneylerde kullanılan küme sisteminin teknik özelliklerini özetlemektedir.

PCLPSO'nun performansı Çizelge 5.2'de listelenen literatürdeki PSO tabanlı dokuz algoritmanın performansı ile karşılaştırılmaktadır. Çizelge 5.5, Çizelge 5.6, Çizelge 5.7 ve Çizelge 5.8'de gösterilen bu dokuz algoritmanın sonuçları (Liang ve ark., 2006)'den alınmıştır.

**Çizelge 5.1.** PCLPSO'nun geliştirildiği küme sisteminin teknik özellikleri

| Özellik | Değeri |
|---|---|
| İşletim sistemi | Windows XP |
| İşlemci | Pentium i5 3.10 GHz |
| Hafıza | 2 GB |
| Java versiyonu | Java SE 1.7 |
| Jade versiyonu | Jade 4.2 |
| Ağ | Gigabit Ethernet |

**Çizelge 5.2.** PCLPSO'yu kıyaslamak için kullanılan algoritmalar

| Kısa adı | Uzun adı | Referansı |
|---|---|---|
| CLPSO | Kapsamlı öğrenme parçacık sürü optimizasyonu | (Liang ve ark., 2006) |
| UPSO | Birleştirilmiş parçacık sürü optimizasyonu | (Parsopoulos ve Vrahatis, 2005) |
| CPSO-H | Kooperatif parçacık sürü optimizasyonu | (Van den Bergh ve Engelbrecht, 2004) |
| FIPS | Tamamen bilgilendirilmiş parçacık sürüsü | (Mendes ve ark., 2004) |
| FDR-PSO | Uygunluk mesafe oranına dayalı parçacık sürüsü optimizasyonu | (Peram ve ark., 2003) |
| PSO-cf | Daralma katsayılı parçacık sürü optimizasyonu | (Clerc ve Kennedy, 2002) |
| PSO-w-local | Eylemsizlik ağırlıklı parçacık sürü optimizasyonun lokal versiyonu | (Clerc ve Kennedy, 2002) |
| PSO-cf-local | Daralma katsayılı parçacık sürü optimizasyonun lokal versiyonu | (Kennedy ve Mendes, 2002) |
| PSO-w | Eylemsizlik ağırlıklı parçacık sürü optimizasyonu | (Shi ve Eberhart, 1998) |

Deneyler fonksiyonların boyutlarına göre üç gruba bölünmüştür ve bu grupların şartları aşağıda açıklanmaktadır. Bütün testleri diğer algoritmalar ile eşit şartlarda yapmak için popülasyon boyutu ve maksimum uygunluk fonksiyonu değerlendirmesi (FE) bütün algoritmalarda aynı sayıda tutulmaktadır.

İlk deney grubunda, fonksiyonlar 10 boyutludur ve PCLPSO algoritmasında bir sunucu ve bir istemci olmak üzere iki sürü kullanılmaktadır. Bu deney grubunda, diğer algoritmaların popülasyon boyutu 10 olduğu için PCLPSO algoritmasındaki her bir sürü 5 parçacığa sahiptir. FE 30,000 olarak ayarlanmaktadır.

İkinci deney grubunda, fonksiyonlar 30 boyutludur ve PCLPSO algoritmasında bir sunucu ve üç istemci olmak üzere dört sürü kullanılmaktadır. Bu deney grubunda, diğer algoritmaların popülasyon boyutu 40 olduğu için PCLPSO algoritmasındaki her bir sürü 10 parçacığa sahiptir. FE 200,000 olarak ayarlanmaktadır.

Üçüncü deney grubunda, fonksiyonlar 100 boyutludur ve PCLPSO algoritmasında bir sunucu ve üç istemci olmak üzere dört sürü kullanılmaktadır. Bu deney grubunda, diğer algoritmaların popülasyon boyutu 60 olduğu için PCLPSO algoritmasındaki her bir sürü 15 parçacığa sahiptir. FE 300,000 olarak ayarlanmaktadır.

Deneylerde kullanılan maksimum uygunluk fonksiyonu değerlendirme sayısı, fonksiyonların boyutları ve algoritmaların popülasyon boyutu bilgileri Çizelge 5.3'de sunulmaktadır. Parantez içindeki sayılar PCLPSO algoritmasındaki sürülerin sayısını belirtmektedir. Eylemsizlik ağırlığı *w*, ivmelenme katsayısı *c* ve boşluk tazeleme *m* değerleri CLPSO algoritmasındaki değerler ile aynıdır. Yani, eylemsizlik ağırlığı *w* iterasyonlar boyunca 0.9'dan 0.2'ye doğrusal olarak düşmektedir, ivmelenme katsayısı *c* 1.49445'e eşittir ve boşluk tazeleme *m* 5'e eşittir. Her bir fonksiyon için 30 bağımsız çalıştırma uygulanmaktadır ve sonuçların ortalama değerleri ve standart sapması Çizelge 5.5, Çizelge 5.6, Çizelge 5.7, Çizelge 5.8 ve Çizelge 5.9'de sunulmaktadır. Küme 10 boyutlu problemler için iki bilgisayara, 30 ve 100 boyutlu problemler için ise dört bilgisayara sahiptir.

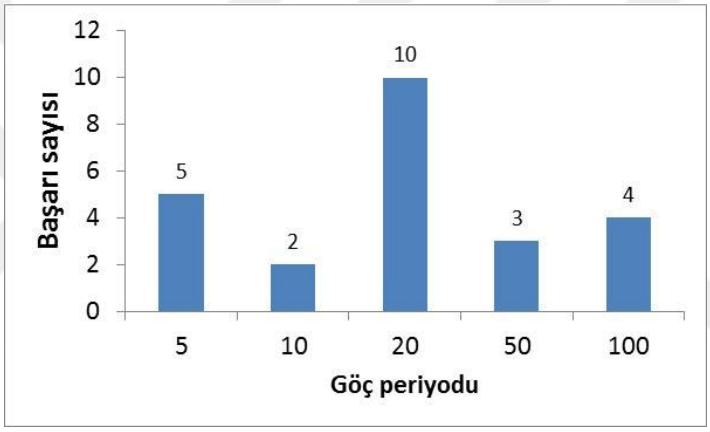
**Çizelge 5.3.** Deneylerdeki fonksiyon boyutu, FE ve popülasyon boyutu

| Deney Grubu | Boyut | FE | PCLPSO popülasyonu | Diğer PSO'ların popülasyonu |
|---|---|---|---|---|
| 1 | 10 | $3x10^4$ | 5x(2) | 10 |
| 2 | 30 | $2x10^5$ | 10x(4) | 40 |
| 3 | 100 | $3x10^5$ | 15x(4) | 60 |

Uygun bir göç periyod *P*'yi belirlemek için, 10-B problemler üzerinde deneyler gerçekleştirilmiştir. Göç periyodları 5, 10, 20, 50 ve 100 olarak alınmaktadır ve elde edilen sonuçlar Çizelge 5.4'de verilmektedir. Çizelgedeki koyu metinler en iyi sonuçları temsil etmektedir. Göç periyodlarını ve bu periyodlarda elde edilen en iyi sonuçların toplam adedini gösteren grafik Şekil 5.1'de sunulmaktadır. Sonuçlara göre, *P* değeri 20'ye eşit olduğu zaman daha iyi çözümler elde edilmiştir. Bundan dolayı, göç periyodu *P* bütün deney gruplardaki test fonksiyonların çözümünde 20 olarak alınmaktadır.

**Çizelge 5.4.** 10-B problemler için farklı göç periyodları altında PCLPSO'nun sonuçları

| $f$ | 5 | 10 | 20 | 50 | 100 |
|---|---|---|---|---|---|
| $f_1$ | **1.1e-60** | 3.9e-53 | 4.3e-57 | 3.6e-32 | 4.7e-34 |
| $f_2$ | **4.6e-01** | 6.9e-01 | 5.8e-01 | 1.1e+00 | 1.3e+00 |
| $f_3$ | **4.0e-15** | **4.0e-15** | **4.0e-15** | **4.0e-15** | **4.0e-15** |
| $f_4$ | 2.2e-03 | 3.7e-07 | 6.6e-04 | 2.5e-03 | **4.5e-08** |
| $f_5$ | **0** | **0** | **0** | **0** | **0** |
| $f_6$ | 3.3e-02 | 3.3e-02 | **0** | **0** | **0** |
| $f_7$ | 1.3e-01 | 3.3e-02 | **0** | 3.3e-02 | 3.3e-02 |
| $f_8$ | 1.5e+01 | 3.9e+00 | **1.3e-04** | 7.9e+00 | 7.9e+00 |
| $f_9$ | **1.5e-10** | 1.6e-09 | 8.3e-09 | 2.9e-09 | 7.8e-09 |
| $f_{10}$ | 8.7e-03 | 1.4e-02 | **5.5e-03** | 2.8e-02 | 1.8e-02 |
| $f_{11}$ | 1.1e-01 | 2.5e-01 | **2.8e-02** | 2.8e-01 | 1.2e-01 |
| $f_{12}$ | 9.6e+00 | 1.0e+01 | **2.6e+00** | 4.7e+00 | 5.1e+00 |
| $f_{13}$ | 6.7e+00 | 6.7e+00 | **3.3e+00** | 5.7e+00 | 6.9e+00 |
| $f_{14}$ | 7.9e+02 | 7.5e+02 | **6.5e+02** | 7.9e+02 | 7.9e+02 |




**Şekil 5.1.** Farklı göç periyodlarında elde edilen en iyi sonuçların toplam adedi

### 5.1.1. 10-B problemlerin sonuçları

Çizelge 5.5 ve Çizelge 5.6 10 algoritma için 10 boyutlu her bir fonksiyonun 30 bağımsız çalıştırılmasından elde edilen sonuçların ortalamalarını ve standart sapmalarını sunmaktadır. Çizelge 5.5 ve Çizelge 5.6'deki koyu metinler en iyi sonuçları temsil etmektedirler. CLPSO algoritmasının tek modlu problemler üzerindeki performansı zayıftır (Liang ve ark., 2006). Fakat sonuçlara göre, PCLPSO algoritması tek modlu problemleri üzerinde CLPSO algoritmasından daha iyi performans göstermektedir ve hatta $f_2$ fonksiyonu üzerinde 10 algoritma arasında en iyi sonuca sahiptir. PCLPSO $f_4$, $f_{10}$, $f_{12}$ ve $f_{13}$ fonksiyonlarında diğer algoritmalardan daha iyi sonuç vermiştir. Bunun yanında, PCLPSO algoritması CLPSO gibi $f_5$, $f_6$ ve $f_7$ fonksiyonlarında global optimumu

bulmaktadır. Ayrıca, PCLPSO $f_3$ ve $f_8$ fonksiyonlarında ikinci en iyi sonuçlara sahiptir. CPSO-H $f_6$ fonksiyonunda global optimumu bulmaktadır. UPSO $f_1$ üzerinde ve FIPS $f_3$, $f_9$ ve $f_{11}$ üzerinde en iyi sonuçlara sahiptir.

Açıktır ki PCLPSO tek modlu ve döndürülmemiş çok modlu fonksiyonlarda mükemmel bir performansa sahiptir ve döndürülmüş çok modlu fonksiyonlarda da iyi bir performans göstermektedir. PCLPSO ve CLPSO'nun sonuçları karşılaştırıldığı zaman, PCLPSO'nun $f_8$, $f_{11}$ ve $f_{14}$ hariç bütün fonksiyonlarda CLPSO'dan daha iyi olduğu açıkça görülmektedir. Bundan dolayı, PCLPSO 10 boyutlu problemler için global aramada CLPSO gibi iyi bir yeteneğe sahiptir.

**Çizelge 5.5.** 10-B problemler için $f_1$-$f_7$ fonksiyon değerlerin ortalamaları ve standart sapmaları

| | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ | $f_7$ |
|---|---|---|---|---|---|---|---|
| PSO-w | 7.96e-51 | 3.08e+00 | 1.58e-14 | 9.69e-02 | 2.28e-03 | 5.82e+00 | 4.05e+00 |
| | 3.56e-50 | 7.69e-01 | 1.60e-14 | 5.01e-02 | 7.04e-03 | 2.96e+00 | 2.58e+00 |
| PSO-cf | 9.84e-105 | 6.98e-01 | 9.18e-01 | 1.19e-01 | 6.69e-01 | 1.25e+01 | 1.20e+01 |
| | 4.21e-104 | 1.46e+00 | 1.01e+00 | 7.11e-02 | 7.17e-01 | 5.17e+00 | 4.99e+00 |
| PSO-w-local | 2.13e-35 | 3.92e+00 | 6.04e-15 | 7.80e-02 | 1.41e-06 | 3.88e+00 | 4.77e+00 |
| | 6.17e-35 | 1.19e+00 | 1.67e-15 | 3.79e-02 | 6.31e-06 | 2.30e+00 | 2.84e+00 |
| PSO-cf-local | 1.37e-79 | 8.60e-01 | 5.78e-02 | 2.80e-02 | 7.85e-02 | 9.05e+00 | 5.95e+00 |
| | 5.60e-79 | 1.56e+00 | 2.58e-01 | 6.34e-02 | 5.16e-02 | 3.48e+00 | 2.60e+00 |
| UPSO | **9.84e-118** | 1.40e+00 | 1.33e+00 | 1.04e-01 | 1.14e+00 | 1.17e+01 | 5.85e+00 |
| | **3.56e-117** | 1.88e+00 | 1.48e+00 | 7.10e-02 | 1.17e+00 | 6.11e+00 | 3.15e+00 |
| FDR-PSO | 2.21e-90 | 8.67e-01 | 3.18e-14 | 9.24e-02 | 3.01e-03 | 7.51e+00 | 3.35e+00 |
| | 9.88e-90 | 1.63e+00 | 6.40e-14 | 5.61e-02 | 7.20e-03 | 3.05e+00 | 2.01e+00 |
| FIPS | 3.15e-30 | 2.78e+00 | **3.75e-15** | 1.31e-01 | 2.02e-03 | 2.12e+00 | 4.35e+00 |
| | 4.56e-30 | 2.26e-01 | **2.13e-14** | 9.32e-02 | 6.40e-03 | 1.33e+00 | 2.80e+00 |
| CPSO-H | 4.98e-45 | 1.53e+00 | 1.49e-14 | 4.07e-02 | 1.07e-15 | **0** | 2.00e-01 |
| | 1.00e-44 | 1.70e+00 | 6.97e-15 | 2.80e-02 | 1.67e-15 | **0** | 4.10e-01 |
| CLPSO | 5.15e-29 | 2.46e+00 | 4.32e-14 | 4.56e-03 | **0** | **0** | **0** |
| | 2.16e-28 | 1.70e+00 | 2.55e-14 | 4.81e-03 | **0** | **0** | **0** |
| PCLPSO | 4.29e-57 | **5.84e-01** | 4.00e-15 | **6.59e-04** | **0** | **0** | **0** |
| (önerilen metod) | 1.17e-56 | **1.59e+00** | 2.41e-30 | **2.50e-03** | **0** | **0** | **0** |

**Çizelge 5.6.** 10-B problemler için $f_8$-$f_{14}$ fonksiyon değerlerin ortalamaları ve standart sapmaları

| | $f_8$ | $f_9$ | $f_{10}$ | $f_{11}$ | $f_{12}$ | $f_{13}$ | $f_{14}$ |
|---|---|---|---|---|---|---|---|
| PSO-w | 3.20e+02 | 2.80e-01 | 1.64e-01 | 6.66e-01 | 9.90e+00 | 1.02e+01 | 5.69e+02 |
| | 1.85e+02 | 5.86e-01 | 9.40e-02 | 7.12e-01 | 3.76e+00 | 3.58e+00 | 2.16e+02 |
| PSO-cf | 9.87e+02 | 1.19e+00 | 1.38e-01 | 2.17e+00 | 1.44e+01 | 1.53e+01 | 1.19e+03 |
| | 2.76e+02 | 1.13e+00 | 1.07e-01 | 1.30e+00 | 6.04e+00 | 6.38e+00 | 4.23e+02 |
| PSO-w-local | 3.26e+02 | 6.39e-15 | 8.04e-02 | 2.14e-01 | 9.25e+00 | 1.09e+01 | 4.72e+02 |
| | 1.32e+02 | 3.18e-15 | 4.46e-02 | 3.65e-01 | 2.74e+00 | 4.08e+00 | 3.07e+02 |
| PSO-cf-local | 8.78e+02 | 2.56e-01 | 7.90e-02 | 1.20e+00 | 1.35e+01 | 1.07e+01 | 9.09e+02 |
| | 2.93e+02 | 5.33e-01 | 5.55e-02 | 1.22e+00 | 6.81e+00 | 2.81e+00 | 3.25e+02 |
| UPSO | 1.08e+03 | 1.00e+00 | 7.76e-02 | 2.61e+00 | 1.52e+01 | 1.47e+01 | 1.27e+03 |
| | 2.68e+02 | 9.27e-01 | 6.40e-02 | 9.48e-01 | 5.25e+00 | 6.53e+00 | 2.29e+02 |
| FDR-PSO | 8.51e+02 | 1.40e-01 | 1.44e-01 | 3.34e-01 | 9.25e+00 | 1.07e+01 | 1.07e+03 |
| | 2.76e+02 | 4.38e-01 | 7.84e-02 | 3.90e-01 | 2.50e+00 | 3.86e+00 | 2.23e+02 |
| FIPS | 7.10e+01 | **2.25e-15** | 1.70e-01 | **5.93e-14** | 1.20e+01 | 8.84e+00 | 2.89e+02 |
| | 1.50e+02 | **1.54e-15** | 1.26e-01 | **1.86e-13** | 6.22e+00 | 3.27e+00 | 2.00e+02 |
| CPSO-H | 2.13e+02 | 1.36e+00 | 1.20e-01 | 4.35e+00 | 2.67e+01 | 1.90e+01 | 9.67e+02 |
| | 1.41e+02 | 8.85e-01 | 8.07e-02 | 1.35e+00 | 1.06e+01 | 9.05e+00 | 3.67e+02 |
| CLPSO | **0** | 3.56e-05 | 4.50e-02 | 3.72e-10 | 5.97e+00 | 5.44e+00 | **1.14e+02** |
| | **0** | 1.57e-04 | 3.08e-02 | 4.40e-10 | 2.88e+00 | 1.39e+00 | **1.28e+02** |
| PCLPSO | 1.27e-04 | 8.28e-09 | **5.50e-03** | 2.75e-02 | **2.61e+00** | **3.26e+00** | 6.48e+02 |
| (önerilen metod) | 2.78e-13 | 4.53e-08 | **1.37e-02** | 4.66e-02 | **4.22e+00** | **3.32e+00** | 2.58e+02 |

### 5.1.2. 30-B problemlerin sonuçları

30 boyutlu her bir fonksiyonun 30 kez bağımsız çalıştırılmasından elde edilen sonuçların ortalamaları ve standart sapmaları Çizelge 5.7 ve Çizelge 5.8'de gösterilmektedir. Çizelge 5.7 ve Çizelge 5.8'deki koyu metinler en iyi sonuçları temsil etmektedir. PCLPSO tek modlu problemler üzerinde iyi bir performansa sahiptir ve hatta $f_2$ üzerinde ikinci en iyi sonuca ulaşmaktadır. Çizelgede görüldüğü gibi, PCLPSO tek modlu problemler üzerinde CLPSO algoritmasından çok daha iyi sonuçlar elde etmektedir.

**Çizelge 5.7.** 30-B problemler için $f_1$-$f_7$ fonksiyon değerlerin ortalamaları ve standart sapmaları

| | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ | $f_7$ |
|---|---|---|---|---|---|---|---|
| PSO-w | 9.78e-30 | 2.93e+01 | 3.94e-14 | 8.13e-03 | 1.30e-04 | 2.90e+01 | 2.97e+01 |
| | 2.50e-29 | 2.51e+01 | 1.12e+00 | 7.16e-03 | 3.30e-04 | 7.70e+00 | 1.39e+01 |
| PSO-cf | 5.88e-100 | 1.11e+01 | 1.12e+00 | 2.06e-02 | 4.10e+00 | 5.62e+01 | 2.85e+01 |
| | 5.40e-100 | 1.81e+00 | 8.65e-01 | 1.90e-02 | 2.20e+00 | 9.76e+00 | 1.14e+01 |
| PSO-w-local | 5.35e-100 | 2.39e+01 | 9.10e-08 | 5.91e-03 | 4.94e-03 | 2.72e+01 | 2.08e+01 |
| | 4.41e-13 | 3.07e+00 | 8.11e-08 | 6.69e-03 | 1.40e-02 | 7.58e+00 | 4.94e+00 |
| PSO-cf-local | 7.70e-54 | 1.71e+01 | 5.33e-15 | 5.91e-03 | 1.16e-01 | 4.53e+01 | 1.54e+01 |
| | 1.59e-53 | 9.16e-01 | 1.87e-15 | 8.70e-03 | 2.79e-01 | 1.17e+01 | 1.67e+01 |
| UPSO | 4.17e-87 | 1.51e+01 | 1.22e-15 | 1.66e-03 | 9.60e+00 | 6.59e+01 | 6.34e+01 |
| | 3.15e-87 | 8.14e-01 | 3.16e-15 | 3.07e-03 | 3.78e+00 | 1.22e+01 | 1.24e+01 |
| FDR-PSO | 4.88e-102 | **5.39e+00** | 2.84e-14 | 1.01e-02 | 7.49e-03 | 2.84e+01 | 1.44e+01 |
| | 1.53e-101 | **1.76e+00** | 4.10e-15 | 1.23e-02 | 1.14e-02 | 8.71e+00 | 6.28e+00 |
| FIPS | 2.69e-12 | 2.45e+01 | 4.81e-07 | 1.16e-06 | 1.54e-01 | 7.30e+01 | 6.08e+01 |
| | 6.84e-13 | 2.19e-01 | 9.17e-08 | 1.87e-06 | 1.48e-01 | 1.24e+01 | 8.35e+00 |
| CPSO-H | **1.16e-113** | 7.08e+00 | 4.93e-14 | 3.63e-02 | 7.82e-15 | **0** | 1.00e-01 |
| | **2.92e-113** | 8.01e+00 | 1.10e-14 | 3.60e-02 | 8.50e-15 | **0** | 3.16e-01 |
| CLPSO | 4.46e-14 | 2.10e+01 | **0** | 3.14e-10 | 3.45e-07 | 4.85e-10 | 4.36e-10 |
| | 1.73e-14 | 2.98e+00 | **0** | 4.64e-10 | 1.94e-07 | 3.63e-10 | 2.44e-10 |
| PCLPSO | 2.87e-28 | 6.63e+00 | 2.05e-14 | **9.22e-12** | **0** | **0** | **0** |
| (önerilen metod) | 6.84e-28 | 1.17e+01 | 6.29e-15 | **4.25e-11** | **0** | **0** | **0** |

**Çizelge 5.8.** 30-B problemler için $f_8$-$f_{14}$ fonksiyon değerlerin ortalamaları ve standart sapmaları

| | $f_8$ | $f_9$ | $f_{10}$ | $f_{11}$ | $f_{12}$ | $f_{13}$ | $f_{14}$ |
|---|---|---|---|---|---|---|---|
| PSO-w | 1.10e+03 | 1.71e+00 | 1.77e-02 | 7.00e+00 | 6.87e+01 | 6.32e+01 | 2.67e+03 |
| | 2.56e+02 | 4.38e-01 | 1.53e-02 | 1.98e+00 | 2.05e+01 | 1.79e+01 | 7.03e+02 |
| PSO-cf | 3.78e+03 | 1.66e+00 | 8.62e-03 | 8.48e+00 | 7.13e+01 | 7.88e+01 | 3.57e+03 |
| | 6.02e+02 | 1.10e+00 | 8.86e-03 | 2.54e+00 | 1.66e+01 | 1.88e+01 | 9.08e+02 |
| PSO-w-local | 1.53e+03 | 5.70e-01 | 1.35e-02 | 5.96e+00 | 4.10e+01 | 5.67e+01 | 2.60e+03 |
| | 3.00e+02 | 7.60e-01 | 1.12e-02 | 2.09e+00 | 7.93e+00 | 1.36e+01 | 5.11e+02 |
| PSO-cf-local | 3.78e+03 | 1.78e-01 | 1.30e-02 | 5.95e+00 | 4.66e+01 | 4.93e+01 | 3.89e+03 |
| | 5.37e+02 | 5.62e-01 | 1.06e-02 | 2.95e+00 | 1.05e+01 | 1.11e+01 | 9.42e+02 |
| UPSO | 4.84e+03 | 2.94e-01 | 1.48e-03 | 1.85e+01 | 7.07e+01 | 7.74e+01 | 5.60e+03 |
| | 4.76e+02 | 6.71e-01 | 3.12e-03 | 3.37e+00 | 1.70e+01 | 1.40e+01 | 6.50e+02 |
| FDR-PSO | 3.61e+03 | 3.59e-01 | 9.60e-03 | 2.50e+00 | 4.44e+01 | 4.36e+01 | 3.78e+03 |
| | 3.06e+02 | 5.93e-01 | 1.24e-02 | 1.46e+00 | 1.37e+01 | 8.96e+00 | 7.59e+02 |
| FIPS | 2.05e+03 | 5.23e-07 | 6.92e-04 | 9.52e-02 | 7.41e+01 | 7.58e+01 | 2.60e+03 |
| | 9.58e+02 | 1.42e-07 | 2.18e-03 | 9.53e-02 | 2.79e+01 | 1.92e+01 | 8.49e+02 |
| CPSO-H | 1.08e+03 | 2.10e+00 | 5.54e-02 | 1.43e+01 | 1.01e+02 | 8.80e+01 | 3.64e+03 |
| | 2.59e+02 | 3.84e-01 | 3.97e-02 | 3.53e+00 | 2.21e+01 | 2.59e+01 | 7.41e+02 |
| CLPSO | **1.27e-12** | 3.43e-04 | **7.04e-10** | 3.07e+00 | 3.46e+01 | **3.77e+01** | **1.70e+03** |
| | **8.79e-13** | 1.91e-04 | **1.25e-11** | 1.61e+00 | 4.59e+00 | **5.56e+00** | **1.86e+02** |
| PCLPSO | 3.82e-04 | **4.03e-12** | 1.19e-05 | **1.15e-08** | **3.60e-07** | 4.24e+01 | 2.84e+03 |
| (önerilen metod) | 7.40e-13 | **1.36e-11** | 4.73e-05 | **4.21e-08** | **1.57e-06** | 9.84e+01 | 3.23e+02 |

Döndürülmemiş çok modlu fonksiyonlar üzerinde, PCLPSO mükemmel bir performans göstermektedir. PCLPSO $f_4$, $f_5$, $f_6$ ve $f_7$ üzerinde diğer algoritmalardan daha iyi bir performans sergilemektedir ve $f_8$ üzerinde ikinci en iyi sonuca ulaşmaktadır. Bu arada, CLPSO sadece $f_3$ ve $f_8$ üzerinde en iyi sonuçlar elde etmektedir. Sonuçlar detaylı analiz edildiği zaman, diğer algoritmaların $f_6$, $f_7$ ve $f_8$ üzerinde birbirine yakın sonuçlar

elde ettiği görülmektedir. Çünkü $f_6$ ve $f_7$ çok sayıda lokal minimuma sahip olduğu için ve $f_8$ derin lokal optimumlara sahip olduğu için diğer algoritmalar lokal optimumlara takılmaktadırlar.

PCLPSO döndürülmüş çok modlu fonksiyonlarda iyi bir performans elde etmektedir. PCLPSO $f_9$, $f_{11}$ ve $f_{12}$ üzerinde 10 algoritma arasında en iyi sonuçları elde etmiştir. Ayrıca $f_{10}$ ve $f_{13}$ üzerinde ikinci en iyi sonuca ulaşmaktadır. CLPSO da döndürülmüş çok modlu fonksiyonlarda iyi bir performans elde etmektedir. CLPSO $f_{10}$, $f_{13}$ ve $f_{14}$ üzerinde en iyi sonuçlar elde etmektedir. Diğer 8 algoritma 10-B problemlerdeki gibi birbirine yakın sonuçlara sahiptir.

### 5.1.3. 100-B problemlerin sonuçları

Bir fonksiyonun boyut sayısı arttıkça fonksiyonun çözümü zorlaşmaktadır. Büyük ölçekli fonksiyonlarda CLPSO ve PCLPSO algoritmalarının performansını ve kararlılığını karşılaştırmak için $f_1$ - $f_{14}$ fonksiyonların boyutları 100 olarak seçilmektedir. Çizelge 5.9 CLPSO ve PCLPSO için her bir fonksiyonun sonuçlarını sunmaktadır. Çizelge 5.9'daki koyu metinler en iyi sonuçları temsil etmektedir. Ayrıca çizelgedeki son sütun, bu iki algoritmanın farklılığını istatistiksel olarak belirleyen Wilcoxon sıra toplam testinin *p* değerlerini göstermektedir. Şekil EK-3.1 CLPSO ve PCLPSO için her bir fonksiyonun sonuçlarını kutu grafiği olarak sunmaktadır. Grafiğin daha anlaşılır olması için dikey eksen logaritmik ölçek ile ölçeklenmiştir.

CLPSO ve PCLPSO algoritmaları aynı problemler üzerinde çalıştırıldığından dolayı ve elde edilen sonuçların dağılımı hakkında yeterli bilgiye sahip olunmadığı için sonuçların kıyaslanmasında parametrik olmayan istatistik testi olan Wilcoxon sıra toplam testi kullanılmıştır. Eğer sonuçların dağılımının normal olduğu bilinseydi parametrik bir test olan öğrenci t-testi kullanılırdı. CLPSO ve PCLPSO algoritmalarının sonuçlarına göre bu iki algoritma arasında 95% ($\alpha=0.05$) önem seviyesinde istatistiksel olarak anlamlı bir fark olup olmadığını belirleyecek olan $H_0$ ve $H_1$ hipotezleri şu şekilde kurulabilir:

$H_0$: CLPSO ve PCLPSO algoritmaları arasında farklılık yoktur.

$H_1$: CLPSO ve PCLPSO algoritmaları birbirinden farklıdır. (çift yönlü test)

Hipotezin kabul edilip edilmeyeceğine Wilcoxon sıra toplam testinin *p* değerine göre karar verilmektedir. Eğer *p* değeri 0.05'den küçük ise o zaman $H_0$ hipotezi 95% önem seviyesinde reddedilir ve $H_1$ hipotezi kabul edilir. Yani, bu iki algoritma 95%

önem seviyesinde birbirinden farklıdır. Tersi durumunda, eğer *p* değeri 0.05'den büyük ise o zaman $H_0$ hipotezi 95% önem seviyesinde kabul edilir ve $H_1$ hipotezi reddedilir. Bu durumda, 95% önem seviyesinde iki algoritma arasında farklılık yoktur. Wilcoxon sıra toplam testi, CLPSO ve PCLPSO algoritmalarının sonuçları üzerinde çalıştırılmıştır. Elde edilen *p* değerleri Çizelge 5.9'da gösterilmektedir. *p* değerlerine göre $H_0$ hipotezi sadece $f_{12}$ ve $f_{14}$ fonksiyonları için kabul edilmektedir, diğer tüm fonksiyonlar için reddedilmektedir. Bu sonuçlardan yola çıkarak, 95% önem seviyesinde istatistiksel olarak CLPSO ve PCLPSO algoritmalarının birbirinden farklı olduğu sonucuna varılmaktadır. $f_1$ - $f_{14}$ fonksiyonları için Wilcoxon sıra toplam testinin hesaplanması Ek-1'de verilmiştir. Testin Z değerinin hesaplanması adımı Çizelge EK-1.4'de, karar modeli adımı grafiksel olarak Şekil EK-1.1 ve Şekil EK-1.2'de ve karar adımı ise Çizelge EK-1.5'de sunulmaktadır.

**Çizelge 5.9.** 100-B problemler için fonksiyon değerlerin ortalamaları ve standart sapmaları

| *f* | CLPSO | PCLPSO | *p* değeri |
|---|---|---|---|
| $f_1$ | 1.39e-004 ± 3.69e-005 | **4.74e-009 ± 3.73e-009** | 0.000 |
| $f_2$ | 9.48e+001 ± 1.71e+001 | **7.29e+001 ± 1.39e+001** | 0.000 |
| $f_3$ | 2.78e-003 ± 6.26e-004 | **1.08e-005 ± 4.10e-006** | 0.000 |
| $f_4$ | 1.36e-004 ± 5.49e-005 | **1.73e-007 ± 5.88e-007** | 0.000 |
| $f_5$ | 5.04e+000 ± 1.52e+000 | **7.59e-004 ± 2.86e-004** | 0.000 |
| $f_6$ | 1.60e+001 ± 4.16e+000 | **1.05e-003 ± 1.72e-003** | 0.000 |
| $f_7$ | **2.02e+000 ± 1.40e+000** | 3.75e+002 ± 1.41e+001 | 0.005 |
| $f_8$ | 3.95e+001 ± 7.10e+001 | **1.27e-003 ± 2.29e-004** | 0.000 |
| $f_9$ | 2.71e-003 ± 6.01e-004 | **2.22e-005 ± 1.17e-005** | 0.000 |
| $f_{10}$ | 3.43e-003 ± 1.66e-003 | **5.43e-005 ± 1.43e-004** | 0.000 |
| $f_{11}$ | 4.39e+000 ± 1.30e+000 | **7.16e-001 ± 8.53e-001** | 0.000 |
| $f_{12}$ | 4.93e+002 ± 9.22e+001 | **4.85e+002 ± 9.11e+001** | 0.147 |
| $f_{13}$ | **3.79e+002 ± 7.42e+001** | 4.42e+002 ± 8.38e+001 | 0.000 |
| $f_{14}$ | **1.84e+004 ± 3.37e+003** | **1.84e+004 ± 3.48e+003** | 0.441 |

Şekil 5.2 ve Şekil 5.3 her bir test fonksiyonu için her bir algoritmanın ortanca koşmasının en iyi uygunluk değeri açısından yakınsama karakteristiğini sunmaktadır. Tek modlu fonksiyonlarda PCLPSO ve CLPSO karşılaştırıldığı zaman, PCLPSO algoritmasının CLPSO algoritmasından daha iyi performans gösterdiği açıktır. $f_1$ ve $f_2$ fonksiyonlarının yakınsama grafikleri göstermektedir ki PCLPSO daha hızlı yakınsamaktadır ve global arama kabiliyetini başarılı bir şekilde devam ettirmektedir. PCLPSO, $f_1$'den daha zor olan $f_2$ fonksiyonunda da CLPSO'dan daha iyi performans göstermektedir. Çünkü PCLPSO lokal çözümlerden daha iyi kaçmaktadır.

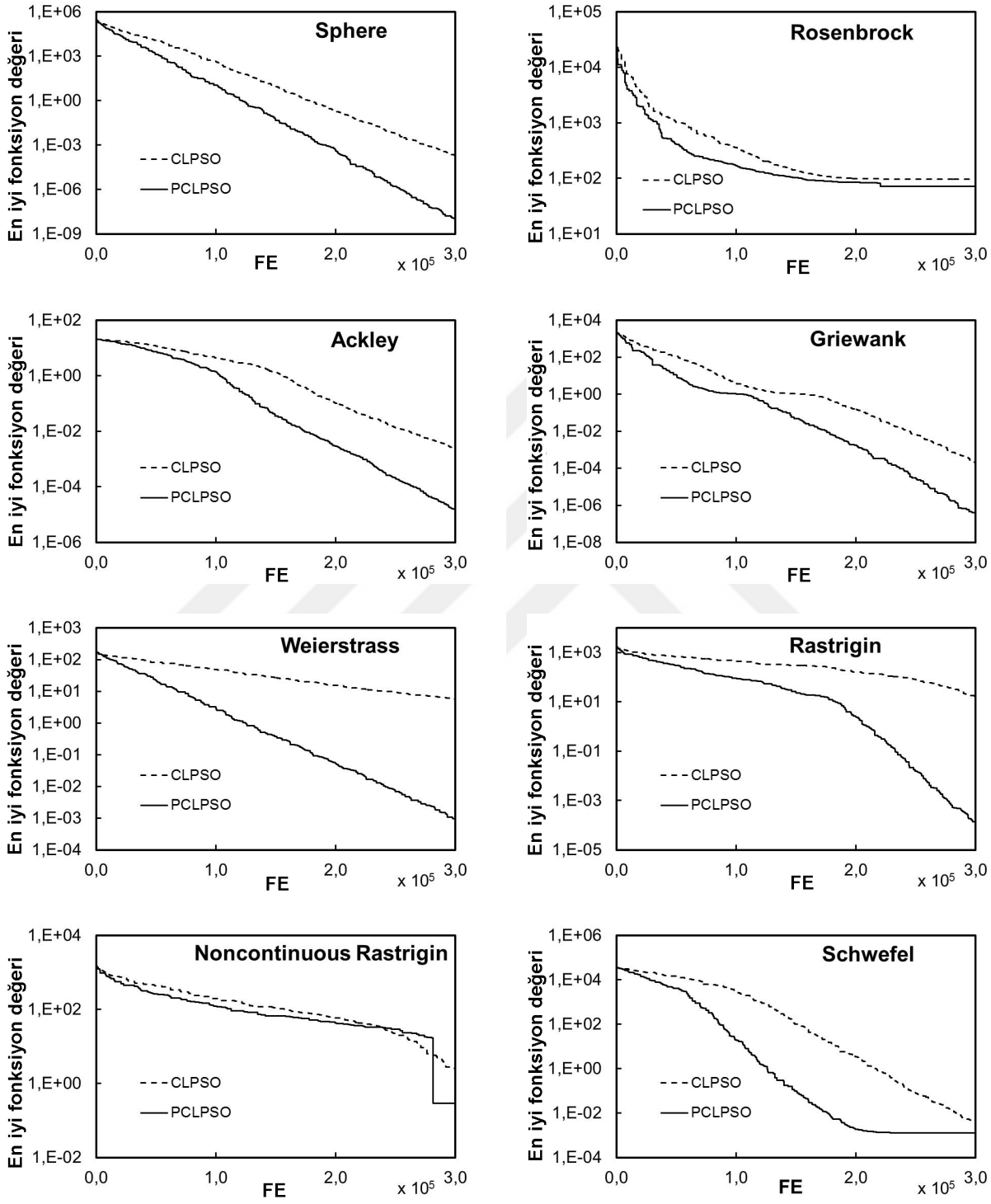
Çok modlu fonksiyonlar kıyaslama fonksiyonlarının zor bir sınıfı olarak kabul edilmektedir. Çünkü fonksiyonun boyut sayısı arttıkça bu fonksiyonun lokal minimum

sayısı üstel olarak artmaktadır. $f_7$ hariç $f_3$ - $f_8$ döndürülmemiş çok modlu fonksiyonlarda, PCLPSO'nun sonuçları CLPSO'nun sonuçlarından daha iyidir. $f_3$ - $f_8$'in yakınsama grafikleri analiz edildiği zaman, PCLPSO lokal optimumlardan başarılı şekilde kurtulmaktadır ve iterasyonlar boyunca global optimuma doğru hareket etmektedir. Özellikle, $f_5$ ve $f_6$'da global optimuma doğru hızlıca yakınsamaktadır.

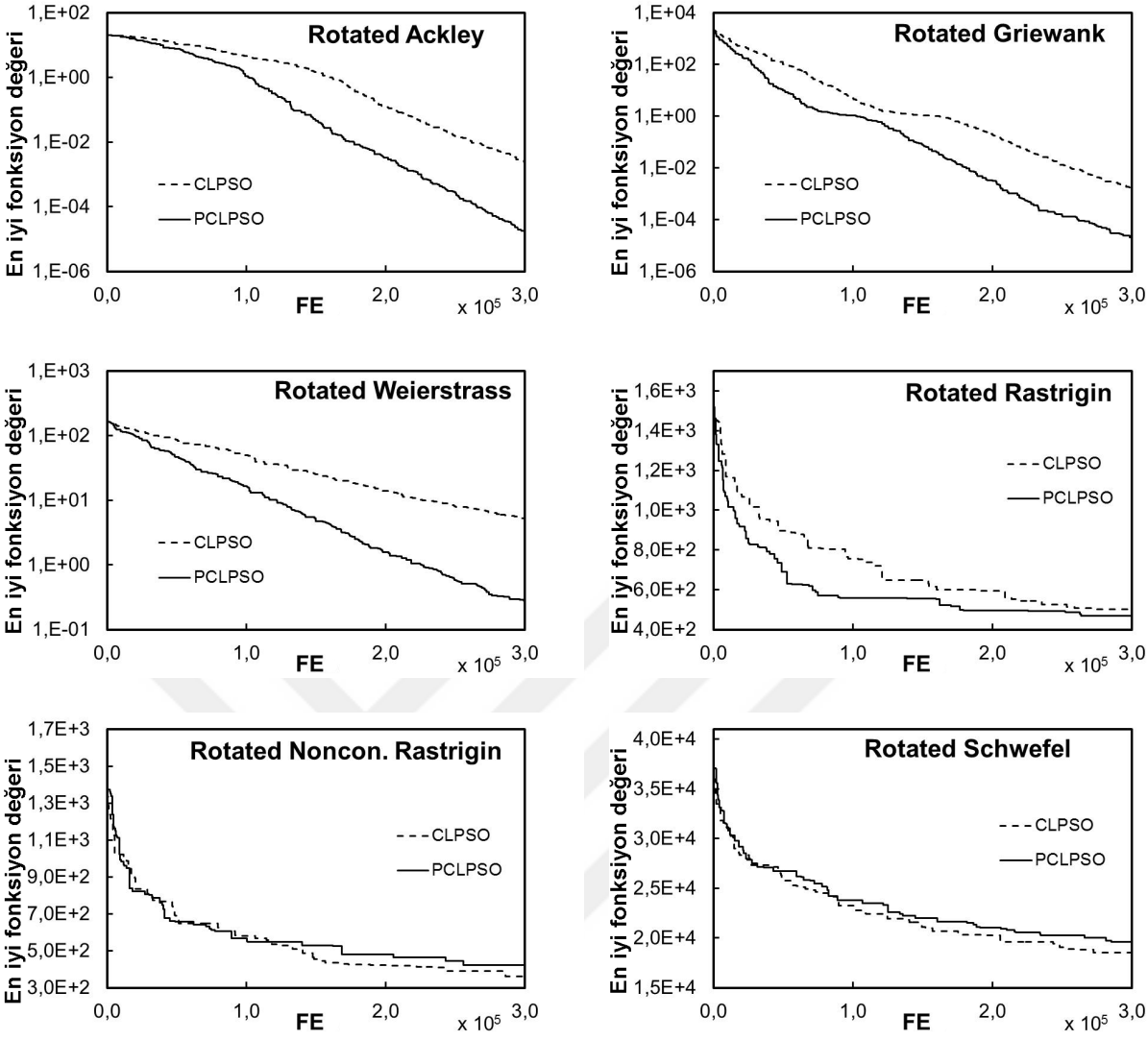
Döndürülmüş çok modlu fonksiyonlarda PCLPSO ve CLPSO karşılaştırıldığı zaman, PCLPSO $f_9$ - $f_{12}$'de CLPSO'dan daha iyi performans göstermektedir. Bu arada, çizelgeden ve grafiklerden görüldüğü gibi her iki algoritma $f_{13}$ ve $f_{14}$'de birbirine yakın sonuçlara sahiptir. $f_9$ - $f_{12}$ fonksiyonlarının yakınsama grafikleri göstermektedir ki PCLPSO daha hızlı yakınsamaktadır ve global arama kabiliyetini başarılı bir şekilde korumaktadır.

Bir diğer önemli husus, bir fonksiyonun boyut sayısı arttıkça bu fonksiyonu hesaplama süresi de doğal olarak artmaktadır. Bu yüzden, büyük ölçekli problemlerin makul bir süre içerisinde çözümü çok önemli hale gelmektedir. Çizelge 5.10, Çizelge 5.11 ve Çizelge 5.12 bu iki algoritmanın 10, 30 ve 100 boyutlu fonksiyonları üzerinde milisaniye cinsinden ortalama hesaplama sürelerini ve PCLPSO'nun hızlanma (*speedup*) değerini sunmaktadır. Çizelge 5.10, Çizelge 5.11 ve Çizelge 5.12'deki koyu metin, CLPSO ve PCLPSO arasındaki hesaplama sürelerinin daha iyi olanını temsil etmektedir. Hızlanma; paralel algoritmaların ölçeklenebilirliğini değerlendirmek için bir performans göstergesidir. Hızlanma, aynı problem üzerinde seri algoritmanın hesaplama zamanının paralel algoritmanın hesaplama zamanına bölünmesi olarak tanımlanmıştır (Alba, 2005). Bu durumda PCLPSO algoritmasının hızlanması CLPSO'nun hesaplama zamanının PCLPSO'nun hesaplama zamanına bölünmesi ile bulunmaktadır. Ayrıca, hızlanmanın grafiksel sunumu Şekil 5.4'te gösterilmektedir. Hızlanma değeri, 1'in üstünde ise PCLPSO algoritması problemi CLPSO'dan daha kısa sürede çözmektedir. Şekil 5.4 ile Çizelge 5.10, Çizelge 5.11 ve Çizelge 5.12'deki sonuçlardan görüldüğü gibi, boyut sayısı arttıkça PCLPSO problemleri CLPSO'dan oldukça daha hızlı çözmektedir. 10-B problemler üzerinde CLPSO'nun hesaplama süreleri PCLPSO'nun hesaplama sürelerinden daha kısadır. Bunun temel nedeni dağıtık sistem üzerinde çalışan PCLPSO algoritmasının sürüleri arasındaki haberleşmenin getirdiği yüktür ki Çizelge 5.18 bunu desteklemektedir. Çizelge 5.18 PCLPSO algoritmasının seri versiyonu ile 10-B problemler üzerinde hesaplama süreleri açısından karşılaştırmasını sunmaktadır. Sürüler arasındaki haberleşmenin getirdiği yükten dolayı, PCLPSO

algoritması 10-B problemler üzerinde $f_5$ ve $f_{11}$ hariç diğer bütün fonksiyonları seri versiyonundan daha fazla sürede çözmektedir.




**Şekil 5.2.** 100-B $f_1$-$f_8$ fonksiyonlarının ortanca koşmasının yakınsama karakteristiği




**Şekil 5.3.** 100-B $f_9$-$f_{14}$ fonksiyonlarının ortanca koşmasının yakınsama karakteristiği

**Çizelge 5.10.** 10-B problemler için CLPSO ve PCLPSO'nun ortalama hesaplama süreleri (ms.) ve PCLPSO hızlanması

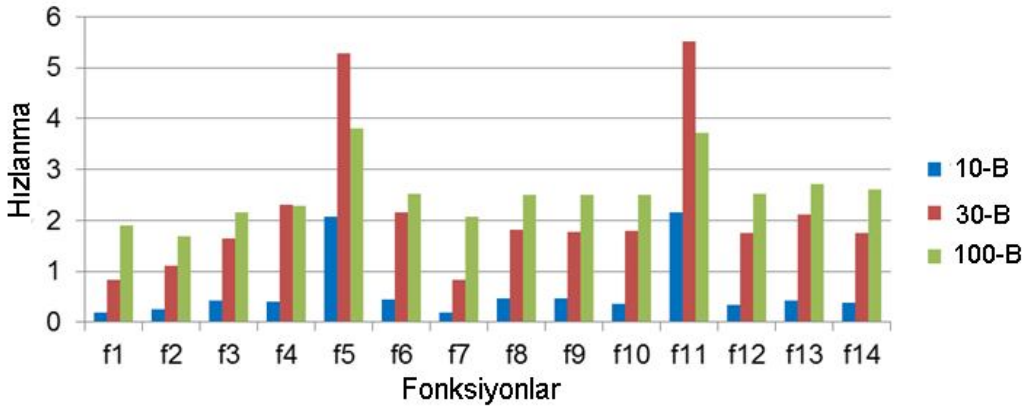
| $f$ | CLPSO | PCLPSO | Hızlanma |
|---|---|---|---|
| $f_1$ | **65** | 340 | 0.19 |
| $f_2$ | **102** | 410 | 0.25 |
| $f_3$ | **182** | 432 | 0.42 |
| $f_4$ | **180** | 446 | 0.40 |
| $f_5$ | 2727 | **1308** | 2.08 |
| $f_6$ | **161** | 367 | 0.44 |
| $f_7$ | **56** | 313 | 0.18 |
| $f_8$ | **167** | 366 | 0.46 |
| $f_9$ | **180** | 384 | 0.47 |
| $f_{10}$ | **144** | 394 | 0.37 |
| $f_{11}$ | 2655 | **1227** | 2.16 |
| $f_{12}$ | **125** | 353 | 0.35 |
| $f_{13}$ | **158** | 376 | 0.42 |
| $f_{14}$ | **136** | 352 | 0.39 |

**Çizelge 5.11.** 30-B problemler için CLPSO ve PCLPSO'nun ortalama hesaplama süreleri (ms.) ve PCLPSO hızlanması

| $f$ | CLPSO | PCLPSO | Hızlanma |
|---|---|---|---|
| $f_1$ | **733** | 870 | 0.84 |
| $f_2$ | 1067 | **959** | 1.11 |
| $f_3$ | 1727 | **1054** | 1.64 |
| $f_4$ | 2564 | **1112** | 2.31 |
| $f_5$ | 45256 | **8578** | 5.28 |
| $f_6$ | 2199 | **1019** | 2.16 |
| $f_7$ | **735** | 879 | 0.84 |
| $f_8$ | 1938 | **1069** | 1.81 |
| $f_9$ | 2046 | **1147** | 1.78 |
| $f_{10}$ | 2087 | **1159** | 1.80 |
| $f_{11}$ | 45610 | **8256** | 5.52 |
| $f_{12}$ | 1951 | **1118** | 1.75 |
| $f_{13}$ | 2527 | **1194** | 2.12 |
| $f_{14}$ | 2074 | **1185** | 1.75 |

**Çizelge 5.12.** 100-B problemler için CLPSO ve PCLPSO'nun ortalama hesaplama süreleri (ms.) ve PCLPSO hızlanması

| $f$ | CLPSO | PCLPSO | Hızlanma |
|---|---|---|---|
| $f_1$ | 5051 | **2643** | 1.91 |
| $f_2$ | 4349 | **2579** | 1.69 |
| $f_3$ | 5655 | **2615** | 2.16 |
| $f_4$ | 6606 | **2880** | 2.29 |
| $f_5$ | 135848 | **35769** | 3.80 |
| $f_6$ | 6728 | **2666** | 2.52 |
| $f_7$ | 5010 | **2419** | 2.07 |
| $f_8$ | 6437 | **2574** | 2.50 |
| $f_9$ | 10068 | **4030** | 2.50 |
| $f_{10}$ | 10222 | **4105** | 2.49 |
| $f_{11}$ | 139029 | **37357** | 3.72 |
| $f_{12}$ | 10786 | **4286** | 2.52 |
| $f_{13}$ | 11926 | **4393** | 2.71 |
| $f_{14}$ | 12109 | **4638** | 2.61 |




**Şekil 5.4.** 10-B, 30-B ve 100-B problemlerde PCLPSO'nun hızlanması

### 5.1.4. Tartışma

Sonuçların özeti şu şekilde listelenebilir:

- Tek modlu fonksiyonlarda, bütün algoritmalar çok iyi performans göstermiştir.
- Döndürülmemiş çok modlu fonksiyonlarda, PCLPSO, CLPSO, FIPS ve CPSO-H algoritmaları 10 boyutlu problemler için diğer algoritmalardan daha iyi sonuçlar vermiştir. PCLPSO ve CLPSO algoritmaları 30 boyutlu problemler için diğerlerinden daha iyi sonuçlar vermiştir. PCLPSO algoritması 100 boyutlu problemler için CLPSO'dan daha iyi performans göstermiştir.
- Döndürülmüş çok modlu fonksiyonlarda, PCLPSO, CLPSO ve FIPS algoritmaları 10 boyutlu problemler için diğer algoritmalardan daha iyi sonuçlar vermiştir. PCLPSO ve CLPSO algoritmaları 30 boyutlu problemler için diğer algoritmalardan daha iyi sonuçlar vermiştir. PCLPSO algoritması 100 boyutlu problemler için CLPSO'dan daha iyi performans göstermiştir.
- PCLPSO algoritması, boyutu 30 ve daha büyük olan fonksiyonlarda CLPSO algoritmasından daha yüksek verimliliğe sahiptir.
- PCLPSO büyük ölçekli problemlerde çok iyi performans göstermiştir.
- Göç periyodu kısa olduğu zaman PCLPSO tek modlu fonksiyonlarda çok iyi performans göstermiştir. Göç periyodu uzun olduğu zaman PCLPSO çok modlu fonksiyonlarda çok iyi performans göstermiştir.
- Göç periyodu ne kadar uzun olursa PCLPSO'nun hesaplama süresi de o kadar düşük olmaktadır.

PSO algoritmasının özellikleri ve başarılı yönleri (Kennedy ve Eberhart, 1995; Van Den Bergh, 2006)'de gösterilmektedir. Bunlardan bazıları kolay gerçekleştirim, basitlik, geniş yelpazedeki farklı optimizasyon problemlerini çözme ve sadece az sayıda parametreye ihtiyaç duymasıdır. PSO birçok avantaja sahip olmasına rağmen PSO'nun dezavantajı aşırı derecede erken yakınsamasıdır ve bundan dolayı lokal optimumlara takılmasıdır (Liang ve ark., 2006; Van Den Bergh, 2006). Birçok PSO türevi bu kusurun üstesinden gelmek için uğraşmaktadırlar (Peram ve ark., 2003; Mendes ve ark., 2004; Van den Bergh ve Engelbrecht, 2004; Liang ve ark., 2006). Bu kusurun üstesinden gelmek ve performansı artırmak için bu çalışmada çoklu-sürü, işbirliği ve paralellik teknikleri kullanılmıştır. Deneysel sonuçlara göre, PCLPSO algoritması birçok

fonksiyonda diğer PSO türevlerinden daha iyi çözümler elde etmiştir. Ayrıca sadece az sayıdaki fonksiyonda kötü çözümler elde etmiştir ki bu durum *no free lunch teoremine* (Wolpert ve Macready, 1997) göre tamamen normaldir ve kabul edilebilir. *No free lunch* teoremine göre bütün problemleri en iyi şekilde çözecek tek bir optimizasyon algoritmasının olması mümkün değildir. Bir algoritma birçok problemde iyi sonuçlar vermesine rağmen bazı problemlerde kötü sonuçlar elde etmesi normal olarak karşılanmaktadır.

PCLPSO algoritmasının bir diğer avantajı onun yüksek hesaplama verimliliğine sahip olmasıdır. PCLPSO algoritması büyük ölçekli problemleri çözerken dağıtık sistem üzerindeki paralel çalışmasından dolayı arama süresini önemli derecede düşürmüştür. CLPSO, fonksiyonun boyutu arttıkça bu fonksiyonu çözmek için PCLPSO'dan ne yazık ki daha çok hesaplama zamanına ihtiyaç duymaktadır.

PCLPSO algoritmasının performansını etkileyen faktörlerden biri kuşkusuz göç periyodudur. Göç periyodu 20'ye eşit olduğu zaman PCLPSO algoritması kıyaslama fonksiyonlarında daha iyi performans elde etmektedir. Fakat göç periyodu farklı problemler için ince ayar yapılmalıdır.

Sonuç olarak, tez çalışmasında geliştirilen PCLPSO algoritmasının iyi bir performans elde ettiği deneysel çalışmalara dayanarak söylenebilir. Ayrıca PCLPSO algoritması diğer algoritmalarla oldukça rekabetçidir ve büyük ölçekli optimizasyon problemlerini çözen herhangi bir metasezgisel algoritmaya bir alternatif olarak kullanılabilir.

### 5.1.5. Haberleşme topolojilerinin karşılaştırılması

PCLPSO algoritmasının başarısını etkileyen önemli faktörlerden bir tanesi şüphesiz ki kullanılan topolojidir. Topoloji, bilgilerin hangi komşu sürülerin arasında paylaşılacağını belirlemektedir. Tez çalışmasında geliştirilen ABT işbirliği stratejisinin deneysel sonuçları bu bölümde sunulmaktadır. ABT stratejisinin topolojisi ve sonuçların karşılaştırılacağı halka, 2B ağ ve tam çizge topolojileri Şekil 4.3'de gösterilmektedir. Bu topolojiler PCLPSO algoritmasına sırayla uygulanmıştır. Deneyler 14 adet kıyaslama fonksiyonu ($f_1$ - $f_{14}$) üzerinde gerçekleştirilmiştir.

Paralel metasezgisel optimizasyonları genellikle büyük ölçekli problemleri çözmede kullanılmaktadır. Bundan dolayı topolojileri karşılaştırmak için deneylerde her bir test fonksiyonun boyutu 100 olarak ayarlanmıştır. Bütün testleri eşit şartlarda

yapmak için aynı parametre değerleri kullanılmıştır. Deneylerde kullanılan bu parametre değerleri Çizelge 5.13'de sunulmaktadır. Deneylerde 6 tane sürü kullanılmıştır. Toplam popülasyon boyutu 60 olarak ayarlanmıştır ve böylece her bir sürüde 10 parçacık bulunmaktadır. FE 300,000 olarak ayarlanmıştır. Eylemsizlik ağırlığı *w* iterasyonlar boyunca 0.9'dan 0.2'ye doğrusal olarak düşmektedir, ivmelenme katsayısı *c* 1.49445'e eşittir ve boşluk tazeleme *m* 5'e eşittir. Göç periyodu *P* 20 olarak ayarlanmıştır. Deneyler teknik özellikleri Çizelge 5.1'de sunulan dağıtık sistem kümesi üzerinde gerçekleştirilmiştir. Küme 6 bilgisayara sahiptir. Her bir fonksiyon için 30 bağımsız çalıştırma uygulanmıştır. Ortalama hesaplama süresi (ms) ve sonuçların ortalama değerleri Çizelge 5.14'de sunulmaktadır.

**Çizelge 5.13.** Topoloji karşılaştırmasında kullanılan parametreler

| Parametre | Değeri |
|---|---|
| Fonksiyon boyutu | 100 |
| Popülasyon boyutu | 60 |
| FE | $3x10^5$ |
| Eylemsizlik ağırlığı (*w*) | $0.9 \rightarrow 0.2$ |
| İvmelenme katsayısı (*c*) | 1.49445 |
| Boşluk tazeleme (*m*) | 5 |
| Sürü sayısı (*n*) | 6 |
| Göç periyodu (*P*) | 20 |

Çizelge 5.14'de verilen sonuçlara göre, ABT'nin topolojisi 14 fonksiyonun 10'unda en yüksek ortalama sonuca sahiptir. Geriye kalan 4 fonksiyonda ise en yüksek ortalama sonuca tam çizge topolojisi sahip olmuştur. Aslında Çizelge 5.14 detaylı incelendiğinde ABT ile tam çizge topolojilerinin sonuçlarının birbirine çok yakın olduğu gözükmektedir. Çünkü bu iki topolojide de herhangi bir sürü bilgi paylaştığı zaman diğer sürülerin hepsi bu bilgiye anında sahip olmaktadır. ABT ve tam çizge topolojilerinden sonra en iyi sonuçlara 2B ağ topolojisi sahip olmuştur ve en kötü sonuçlar halka topolojisinde elde edilmiştir. 2B ağ topolojisinde bir sürü sadece kendi komşuları ile haberleşmektedir. Komşusu olmayan sürüden anında bilgi alamamaktadır. Halka topolojisinde ise durum daha da kötüdür ve bir sürünün kendisinden sonraki bir sürüden bilgi alması birkaç göç sürecinden sonra olmaktadır. Bundan dolayı, 4 topoloji arasında en kötü sonuçları halka topolojisi elde etmiştir.

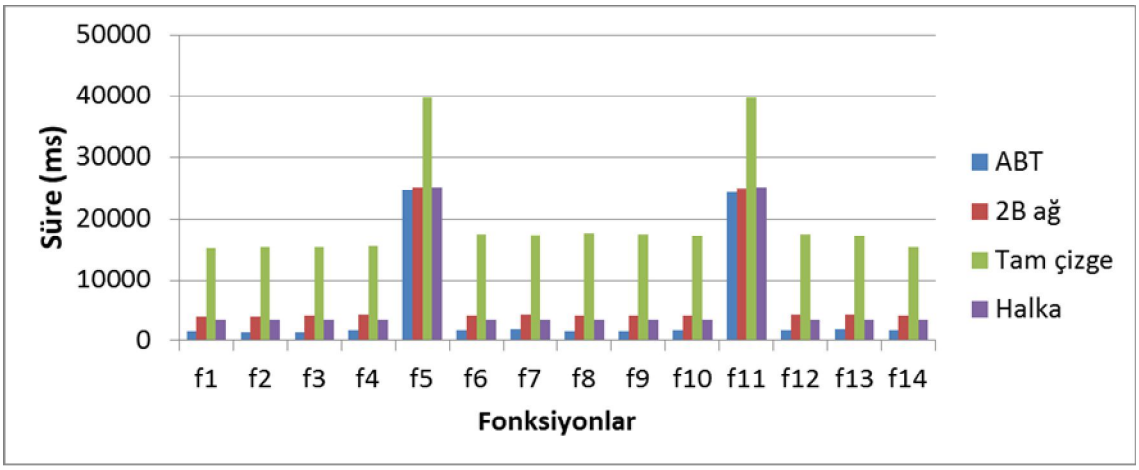
Göç sürecinde eğer bilgiler çok fazla sürü arasında paylaşılırsa algoritmanın hesaplama verimliliği doğal olarak düşmektedir. Eğer bilgiler çok az sürü arasında

paylaşılırsa hesaplama verimliliği artmaktadır. Şekil 4.3'den de görüldüğü gibi ABT topolojisinde haberleşmeler sadece sunucu-sürü ile istemci sürüler arasındadır ve istemci sürüler arasında doğrudan bir haberleşme bulunmamaktadır. Bundan dolayı bu topolojide düşük haberleşme frekansı olduğu için yüksek hesaplama verimliliği elde edilmektedir. Çizelge 5.14 ve Şekil 5.5 hesaplama süresi açısından incelendiğinde, 14 fonksiyon için ABT topolojisinin diğer topolojilerden daha verimli sonuçlar ürettiği görülmektedir. En uzun sürede sonuç üreten topoloji ise tam çizge topolojisidir. Çünkü göç sürecinde bu topolojideki her bir sürü, topolojideki diğer bütün sürüler ile doğrudan bilgi takası yapmaktadır. Grafikten de görüldüğü gibi, 2B ağ ve halka topolojilerinin hesaplama süreleri birbirine yakındır. Bu iki topolojide haberleşme frekansı ABT topolojisinden fazladır. Bundan dolayı bu iki topolojinin hesaplama süresi ABT topolojisinden daha fazladır.

Genel olarak deney sonuçlarına bakıldığında hem hesaplama süresi hem de çözümlerin kalitesi açısından, ABT'nin topolojisinin diğer topolojilerden daha başarılı sonuçlar ürettiği görülmektedir.

**Çizelge 5.14.** Topolojilerin karşılaştırması

| $f$ | Sonuçların ortalaması | | | | Ortalama hesaplama süresi (ms) | | | |
|---|---|---|---|---|---|---|---|---|
| | ABT | 2B ağ | Tam çizge | Halka | ABT | 2B ağ | Tam çizge | Halka |
| $f_1$ | 3.39E-07 | 3.78E-04 | **1.27E-07** | 2.64E-03 | **1572** | 3970 | 15194 | 3440 |
| $f_2$ | **8.20E+01** | 1.01E+02 | 9.97E+01 | 1.11E+02 | **1333** | 3956 | 15296 | 3429 |
| $f_3$ | **1.78E-05** | 5.17E-01 | 6.31E-05 | 1.39E+00 | **1333** | 4138 | 15423 | 3459 |
| $f_4$ | **2.32E-07** | 1.10E-03 | 8.95E-04 | 4.67E-03 | **1740** | 4269 | 15523 | 3455 |
| $f_5$ | **3.42E-04** | 8.20E-01 | 6.53E-01 | 9.40E-01 | **24626** | 25106 | 39736 | 25252 |
| $f_6$ | 1.08E+00 | 2.50E+01 | **4.43E-01** | 6.07E+01 | **1637** | 4251 | 17459 | 3454 |
| $f_7$ | **1.69E+00** | 4.17E+01 | 4.99E+01 | 6.78E+01 | **1843** | 4326 | 17423 | 3456 |
| $f_8$ | 3.95E+00 | 2.95E+03 | **3.45E-01** | 4.20E+03 | **1562** | 4155 | 17595 | 3435 |
| $f_9$ | **1.47E-05** | 5.66E-01 | 6.25E-04 | 1.50E+00 | **1554** | 4138 | 17543 | 3418 |
| $f_{10}$ | **2.93E-07** | 2.94E-04 | 1.41E-05 | 6.48E-03 | **1701** | 4232 | 17196 | 3467 |
| $f_{11}$ | **4.03E-04** | 8.60E-01 | 7.28E-04 | 1.95E+00 | **24394** | 24969 | 39700 | 25222 |
| $f_{12}$ | **1.06E+00** | 3.08E+01 | 4.17E+00 | 5.88E+01 | **1710** | 4302 | 17556 | 3444 |
| $f_{13}$ | **2.17E+00** | 4.11E+01 | 5.15E+00 | 6.81E+01 | **1880** | 4339 | 17213 | 3485 |
| $f_{14}$ | 1.58E+01 | 3.32E+03 | **3.20E+00** | 4.69E+03 | **1588** | 4189 | 15380 | 3439 |




**Şekil 5.5.** Topolojilerin süre açısından karşılaştırılması

### 5.1.6. PCLPSO algoritmasının seri versiyonu ile karşılaştırılması

Önceki bölümde PCLPSO algoritması, bu algoritmanın geleneksel seri versiyonu olan CLPSO algoritması ile karşılaştırılmıştır. Fakat açıkça görülmektedir ki bu iki algoritma farklı algoritmalardır. Daha adil bir karşılaştırma için, Alba (2005) paralel bir algoritmayı tek bir işlemci üzerinde çalıştırarak seri halinin elde edildiğini ve bu iki algoritmanın karşılaştırılmasının gerektiğini belirtmektedir. Bundan dolayı paralel algoritma olan PCLPSO, tek bir işlemci üzerinde çalıştırılarak PCLPSO algoritmasının seri hali elde edilmektedir. Bu bölümde PCLPSO algoritması seri hali ile karşılaştırılmaktadır.

PCLPSO algoritması ve seri hali aynı problemler üzerinde çalıştırıldığından dolayı ve elde edilen sonuçların dağılımı hakkında yeterli bilgiye sahip olunmadığı için sonuçların kıyaslanmasında parametrik olmayan istatistik testi olan Wilcoxon sıra toplam testi kullanılmıştır. Bu iki algoritmanın sonuçlarına göre bu iki algoritma arasında 95% ($\alpha$=0.05) önem seviyesinde istatistiksel olarak bir fark olup olmadığını belirlemek için $H_0$ ve $H_1$ hipotezleri şu şekilde kurulabilir:

$H_0$: PCLPSO algoritması ve seri hali arasında farklılık yoktur.

$H_1$: PCLPSO algoritması ve seri hali birbirinden farklıdır. (çift yönlü test)

Hipotezin kabul edilip edilmeyeceğine Wilcoxon sıra toplam testinin *p* değerine göre karar verilmektedir. Eğer *p* değeri 0.05'den küçük ise o zaman $H_0$ hipotezi 95% önem seviyesinde reddedilir ve $H_1$ hipotezi kabul edilir. Yani, bu iki algoritma 95% önem seviyesinde birbirinden farklıdır. Tersi durumunda, eğer *p* değeri 0.05'den büyük

ise o zaman $H_0$ hipotezi 95% önem seviyesinde kabul edilir ve $H_1$ hipotezi reddedilir. Bu durumda, 95% önem seviyesinde iki algoritma arasında farklılık yoktur. Wilcoxon sıra toplam testi, PCLPSO algoritmasının ve seri halinin sonuçları üzerinde çalıştırılmıştır. Elde edilen *p* değerleri Çizelge 5.15, Çizelge 5.16 ve Çizelge 5.17'de gösterilmektedir.

10 boyutlu her bir fonksiyonun 30 bağımsız çalıştırılmasından elde edilen sonuçların ortalamaları ve standart sapmaları Çizelge 5.15'de sunulmaktadır. Çizelge 5.15'e göre, PCLPSO algoritması ve seri hali bütün fonksiyonlarda yakın sonuçlar elde etmişlerdir. Ayrıca Çizelge 5.15'deki son sütun, bu iki algoritmanın farklı olup olmadığını istatistiksel olarak belirleyen Wilcoxon sıra toplam testinin *p* değerlerini göstermektedir. *p* değerlerine göre $H_0$ hipotezi sadece $f_{13}$ fonksiyonu için reddedilmektedir, diğer tüm fonksiyonlar için kabul edilmektedir. Bu sonuçlardan yola çıkarak, 95% önem seviyesinde istatistiksel olarak PCLPSO algoritması ve seri hali arasında farklılık olmadığı sonucuna varılmaktadır.

**Çizelge 5.15.** 10-B problemler için fonksiyon değerlerin ortalamaları ve standart sapmaları ve Wilcoxon sıra toplam testine göre *p* değeri

| *f* | Seri | | PCLPSO | | *p* değeri |
|---|---|---|---|---|---|
| | Ortalama | S.S. | Ortalama | S.S. | |
| $f_1$ | 2.16e-57 | 7.69e-57 | 4.29e-57 | 1.17e-56 | 0.894 |
| $f_2$ | 1.72e-01 | 5.91e-01 | 5.84e-01 | 1.59e+00 | 0.888 |
| $f_3$ | 4.00e-15 | 1.60e-30 | 4.00e-15 | 2.41e-30 | 1.000 |
| $f_4$ | 2.47e-04 | 1.35e-03 | 6.59e-04 | 2.50e-03 | 0.240 |
| $f_5$ | 0.00e+00 | 0.00e+00 | 0.00e+00 | 0.00e+00 | 1.000 |
| $f_6$ | 0.00e+00 | 0.00e+00 | 0.00e+00 | 0.00e+00 | 1.000 |
| $f_7$ | 0.00e+00 | 0.00e+00 | 0.00e+00 | 0.00e+00 | 1.000 |
| $f_8$ | 1.27e-04 | 1.66e-13 | 1.27e-04 | 2.78e-13 | 1.000 |
| $f_9$ | 3.82e-11 | 1.83e-10 | 8.28e-09 | 4.53e-08 | 0.966 |
| $f_{10}$ | 1.45e-02 | 2.79e-02 | 5.50e-03 | 1.37e-02 | 0.391 |
| $f_{11}$ | 1.15e-01 | 4.38e-01 | 2.75e-02 | 4.66e-02 | 0.574 |
| $f_{12}$ | 4.16e+00 | 5.18e+00 | 2.61e+00 | 4.22e+00 | 0.065 |
| $f_{13}$ | 5.78e+00 | 2.37e+00 | 3.26e+00 | 3.32e+00 | 0.003 |
| $f_{14}$ | 6.19e+02 | 2.65e+02 | 6.48e+02 | 2.58e+02 | 0.416 |

Çizelge 5.16 30 boyutlu her bir fonksiyonun 30 bağımsız çalıştırılmasından elde edilen sonuçların ortalamalarını ve standart sapmalarını sunmaktadır. Çizelge 5.16'ya göre, PCLPSO algoritması ve seri hali $f_{11}$ ve $f_{12}$ hariç bütün fonksiyonlarda yakın sonuçlar elde etmişlerdir. Ayrıca Çizelge 5.16'daki *p* değerlerine göre $H_0$ hipotezi sadece $f_{11}, f_{12}$ ve $f_{13}$ fonksiyonu için reddedilmektedir, diğer tüm fonksiyonlar için kabul edilmektedir. Bu sonuçlardan yola çıkarak, 95% önem seviyesinde istatistiksel olarak PCLPSO algoritması ve seri hali arasında farklılık olmadığı sonucuna varılmaktadır.

100 boyutlu her bir fonksiyonun 30 bağımsız çalıştırılmasından elde edilen sonuçların ortalamaları ve standart sapmaları Çizelge 5.17'de sunulmaktadır. Çizelge 5.17'ye göre, PCLPSO algoritması ve seri hali bütün fonksiyonlarda yakın sonuçlar elde etmişlerdir. Ayrıca Çizelge 5.17'deki $p$ değerlerine göre $H_0$ hipotezi sadece $f_2$, $f_3$ ve $f_{13}$ fonksiyonu için reddedilmektedir, diğer tüm fonksiyonlar için kabul edilmektedir. Genel olarak sonuçlara bakıldığında, 95% önem seviyesinde istatistiksel olarak PCLPSO algoritması ve PCLPSO algoritmasının seri versiyonu arasında farklılık olmadığı sonucuna varılmaktadır.

**Çizelge 5.16.** 30-B problemler için fonksiyon değerlerin ortalamaları ve standart sapmaları ve Wilcoxon sıra toplam testine göre $p$ değeri

| $f$ | Seri | | PCLPSO | | $p$ değeri |
|---|---|---|---|---|---|
| | Ortalama | S.S. | Ortalama | S.S. | |
| $f_1$ | 7.23e-24 | 3.96e-23 | 2.87e-28 | 6.84e-28 | 0.074 |
| $f_2$ | 6.05e+00 | 9.25e+00 | 6.63e+00 | 1.17e+01 | 0.506 |
| $f_3$ | 3.13e-13 | 1.60e-12 | 2.05e-14 | 6.29e-15 | 0.629 |
| $f_4$ | 2.56e-11 | 1.07e-10 | 9.22e-12 | 4.25e-11 | 0.103 |
| $f_5$ | 0.00e+00 | 0.00e+00 | 0.00e+00 | 0.00e+00 | 1.000 |
| $f_6$ | 0.00e+00 | 0.00e+00 | 0.00e+00 | 0.00e+00 | 1.000 |
| $f_7$ | 0.00e+00 | 0.00e+00 | 0.00e+00 | 0.00e+00 | 1.000 |
| $f_8$ | 3.82e-04 | 2.21e-19 | 3.82e-04 | 7.40e-13 | 1.000 |
| $f_9$ | 1.75e-10 | 8.29e-10 | 4.03e-12 | 1.36e-11 | 0.311 |
| $f_{10}$ | 1.48e-04 | 7.50e-04 | 1.19e-05 | 4.73e-05 | 0.813 |
| $f_{11}$ | 7.93e-01 | 5.91e-01 | 1.15e-08 | 4.21e-08 | 0.000 |
| $f_{12}$ | 5.15e+01 | 1.13e+01 | 3.60e-07 | 1.57e-06 | 0.000 |
| $f_{13}$ | 4.98e+01 | 1.02e+01 | 4.24e+01 | 9.84e+01 | 0.003 |
| $f_{14}$ | 2.73e+03 | 5.43e+02 | 2.84e+03 | 3.23e+02 | 0.574 |

**Çizelge 5.17.** 100-B problemler için fonksiyon değerlerin ortalamaları ve standart sapmaları ve Wilcoxon sıra toplam testine göre $p$ değeri

| $f$ | Seri | | PCLPSO | | $p$ değeri |
|---|---|---|---|---|---|
| | Ortalama | S.S. | Ortalama | S.S. | |
| $f_1$ | 2.37e-09 | 2.65e-09 | 4.74e-09 | 3.73e-09 | 0.081 |
| $f_2$ | 8.55e+01 | 5.73e+00 | 7.29e+01 | 1.39e+01 | 0.000 |
| $f_3$ | 2.22e-05 | 8.04e-05 | 1.08e-05 | 4.10e-06 | 0.021 |
| $f_4$ | 3.54e-07 | 1.50e-06 | 1.73e-07 | 5.88e-07 | 0.156 |
| $f_5$ | 9.59e-04 | 2.19e-03 | 7.59e-04 | 2.86e-04 | 0.062 |
| $f_6$ | 1.22e-03 | 2.09e-03 | 1.05e-03 | 1.72e-03 | 0.318 |
| $f_7$ | 1.13e+01 | 1.31e+01 | 3.75e+02 | 1.41e+01 | 0.953 |
| $f_8$ | 1.27e-03 | 1.72e-08 | 1.27e-03 | 2.29e-04 | 0.767 |
| $f_9$ | 6.04e-05 | 2.47e-04 | 2.22e-05 | 1.17e-05 | 0.069 |
| $f_{10}$ | 1.05e-04 | 3.41e-04 | 5.43e-05 | 1.43e-04 | 0.064 |
| $f_{11}$ | 6.37e-01 | 9.10e-01 | 7.16e-01 | 8.53e-01 | 0.478 |
| $f_{12}$ | 4.99e+02 | 3.70e+01 | 4.85e+02 | 9.11e+01 | 0.280 |
| $f_{13}$ | 4.61e+02 | 4.07e+01 | 4.42e+02 | 8.38e+01 | 0.035 |
| $f_{14}$ | 1.88e+04 | 1.04e+03 | 1.84e+04 | 3.48e+03 | 0.198 |

Çizelge 5.18, Çizelge 5.19 ve Çizelge 5.20 bu iki algoritmanın 10, 30 ve 100 boyutlu fonksiyonları üzerinde milisaniye cinsinden ortalama hesaplama sürelerini ve PCLPSO'nun hızlanma (*speedup*) değerini sunmaktadır.

Hızlanma değeri, 1'in üstünde ise PCLPSO algoritması problemi seri halinden daha kısa sürede çözmektedir. Çizelge 5.18, Çizelge 5.19 ve Çizelge 5.20'deki sonuçlardan görüldüğü gibi, boyut sayısı arttıkça PCLPSO problemleri seri versiyonundan oldukça daha hızlı çözmektedir. 10-B problemler üzerinde seri versiyonun hesaplama süreleri PCLPSO'nun hesaplama sürelerinden daha kısadır. Bunun temel nedeni dağıtık sistem üzerinde çalışan PCLPSO algoritmasının sürüleri arasındaki haberleşmenin getirdiği yüktür.

**Çizelge 5.18.** 10-B problemler için ortalama hesaplama süreleri (ms.) ve PCLPSO hızlanması

| $f$ | Seri | PCLPSO | Hızlanma |
|---|---|---|---|
| $f_1$ | **316** | 340 | 0.93 |
| $f_2$ | **295** | 410 | 0.72 |
| $f_3$ | **315** | 432 | 0.73 |
| $f_4$ | **319** | 446 | 0.72 |
| $f_5$ | 2247 | **1308** | 1.72 |
| $f_6$ | **283** | 367 | 0.77 |
| $f_7$ | **276** | 313 | 0.88 |
| $f_8$ | **316** | 366 | 0.86 |
| $f_9$ | **335** | 384 | 0.87 |
| $f_{10}$ | **325** | 394 | 0.82 |
| $f_{11}$ | 2113 | **1227** | 1.72 |
| $f_{12}$ | **310** | 353 | 0.88 |
| $f_{13}$ | **318** | 376 | 0.85 |
| $f_{14}$ | **314** | 352 | 0.89 |

**Çizelge 5.19.** 30-B problemler için ortalama hesaplama süreleri (ms.) ve PCLPSO hızlanması

| $f$ | Seri | PCLPSO | Hızlanma |
|---|---|---|---|
| $f_1$ | 1715 | **870** | 1.97 |
| $f_2$ | 1668 | **959** | 1.74 |
| $f_3$ | 1963 | **1054** | 1.86 |
| $f_4$ | 2146 | **1112** | 1.93 |
| $f_5$ | 33505 | **8578** | 3.91 |
| $f_6$ | 2018 | **1019** | 1.98 |
| $f_7$ | 2058 | **879** | 2.34 |
| $f_8$ | 2032 | **1069** | 1.90 |
| $f_9$ | 2325 | **1147** | 2.03 |
| $f_{10}$ | 2379 | **1159** | 2.05 |
| $f_{11}$ | 32763 | **8256** | 3.97 |
| $f_{12}$ | 2422 | **1118** | 2.17 |
| $f_{13}$ | 2715 | **1194** | 2.27 |
| $f_{14}$ | 2600 | **1185** | 2.19 |

**Çizelge 5.20.** 100-B problemler için ortalama hesaplama süreleri (ms.) ve PCLPSO hızlanması

| $f$ | Seri | PCLPSO | Hızlanma |
|---|---|---|---|
| $f_1$ | 4306 | **2643** | 1.63 |
| $f_2$ | 4320 | **2579** | 1.68 |
| $f_3$ | 5717 | **2615** | 2.19 |
| $f_4$ | 6690 | **2880** | 2.32 |
| $f_5$ | 146188 | **35769** | 4.09 |
| $f_6$ | 6554 | **2666** | 2.46 |
| $f_7$ | 7129 | **2419** | 2.95 |
| $f_8$ | 5998 | **2574** | 2.33 |
| $f_9$ | 11653 | **4030** | 2.89 |
| $f_{10}$ | 11986 | **4105** | 2.92 |
| $f_{11}$ | 153080 | **37357** | 4.10 |
| $f_{12}$ | 12680 | **4286** | 2.96 |
| $f_{13}$ | 14476 | **4393** | 3.30 |
| $f_{14}$ | 13823 | **4638** | 2.98 |

## 5.2. PACO-3Opt Algoritmasının Deneysel Sonuçları

Bu bölümde, PACO-3Opt algoritmasının deneysel sonuçları sunulmaktadır. Deneylerde kullanılan donanım ve yazılımın özellikleri Çizelge 5.21'de gösterilmektedir. Problemlerin boyutu açısından deneyler iki gruba bölünmüştür: küçük ölçekli ve büyük ölçekli. Algoritmaları karşılaştırmak için, küçük-ölçekli problemlerde 10 farklı simetrik GSP örneği (eil51, berlin52, rat99, eil76, st70, kroA100, lin105, kroA200, ch150 ve eil101) ve büyük-ölçekli problemlerde 11 farklı simetrik GSP örneği (rd400, fl417, pr439, pcb442, d493, u574, rat575, p654, d657, u724, rat783) kullanılmıştır. Bu problemlerin şehir sayısı ve bilinen en iyi tur uzunluğu Çizelge 3.2'de gösterilmektedir. Simetrik GSP'nin tanımı kısaca şu şekildedir: $n$ düğümlü bir küme ve düğümlerin her bir çifti için uzaklıklar göz önüne alındığında, amaç her düğümü bir kereliğine ziyaret eden minimum toplam uzunlukta bir tur bulmaktır. Düğüm $i$'den düğüm $j$'ye olan uzaklık düğüm $j$'den düğüm $i$'ye olan uzaklık ile aynıdır (Reinelt, 1995). Bütün problem örnekleri, GSP kıyaslama örneklerinin herkesçe ulaşılabilen koleksiyonu olan TSPLIB (Reinelt, 1991; 1995) kütüphanesinden alınmıştır. Bu örnekler global optimizasyon topluluğu tarafından iyi bilinmektedir ve optimizasyon algoritmalarının testi için yaygın olarak kullanılırlar. GSP'de şehir sayısı arttıkça optimal turu belirlemek oldukça zorlaşmaktadır.

**Çizelge 5.21.** PACO-3Opt'nun geliştirildiği küme sisteminin teknik özellikleri

| Özellik | Değeri |
|---|---|
| İşletim sistemi | Windows XP |
| Cpu | Pentium i5 3.10 GHz |
| Hafıza | 2 GB |
| Java versiyonu | Java SE 1.7 |
| Jade versiyonu | Jade 4.3 |
| Ağ | Gigabit Ethernet |

### 5.2.1. Küçük-ölçekli problemler üzerinde PACO-3Opt algoritmasının değerlendirilmesi

Bu bölümde küçük-ölçekli problemler olarak sınıflandırdığımız 10 farklı simetrik GSP örneğini kullandık: eil51, berlin52, rat99, eil76, st70, kroA100, lin105, kroA200, ch150 ve eil101. Bu bölümdeki tüm deneylerde, PACO-3Opt algoritmasının parametreleri Çizelge 5.22'deki gibi ayarlanmıştır. Deneyler iki bilgisayara sahip küme üzerinde gerçekleştirilmiştir. PACO-3Opt'da biri sunucu ve biri istemci olan iki koloni kullanılmıştır. Karşılaştırılacak diğer algoritmaların popülasyon boyutu 10 olduğu için PACO-3Opt algoritmasındaki her bir koloni 5 karıncaya sahiptir. Feromon buharlaşma oranı $\rho$ 0.1 olarak ve $Q$ sabiti 1 olarak ayarlanmıştır. Maksimum iterasyon sayısı 1000 olarak ayarlanmıştır. $\alpha$ ve $\beta$ parametreleri için Mahi ve ark. tarafından önerilen değerler kullanılmıştır ve bu değerler Çizelge 5.23'de gösterilmektedir. Çeşitli göç periyodları için her bir GSP örneğinin ortalama tur uzunlukları Çizelge 5.24 ve Şekil 5.6'da gösterilmektedir. Şekil 5.6'dan görüldüğü gibi, eğer göç periyodu uzunsa tur uzunluklarının ortalama maliyeti yükselmektedir. Çizelge 5.24 ve Şekil 5.6'da görüldüğü gibi göç periyodu $P$ 5 civarında iken daha iyi sonuçlar elde edilmektedir. Bu nedenle, göç periyodu $P$ deneylerdeki bütün GSP örnekleri için 5 olarak ayarlanmıştır. Ayrıca, her bir test bağımsız olarak 20 kez tekrarlanmıştır.

**Çizelge 5.22.** Küçük ölçekli GSP örnekleri için parametre değerleri

| Parametre | $n$ | $m$ | $P$ | $\rho$ | $Q$ | maksimum iterasyon |
|---|---|---|---|---|---|---|
| Değeri | 2 | 5 | 5 | 0.1 | 1 | 1000 |

**Çizelge 5.23.** Küçük ölçekli GSP örnekleri için $\alpha$ ve $\beta$ parametrelerinin değerleri

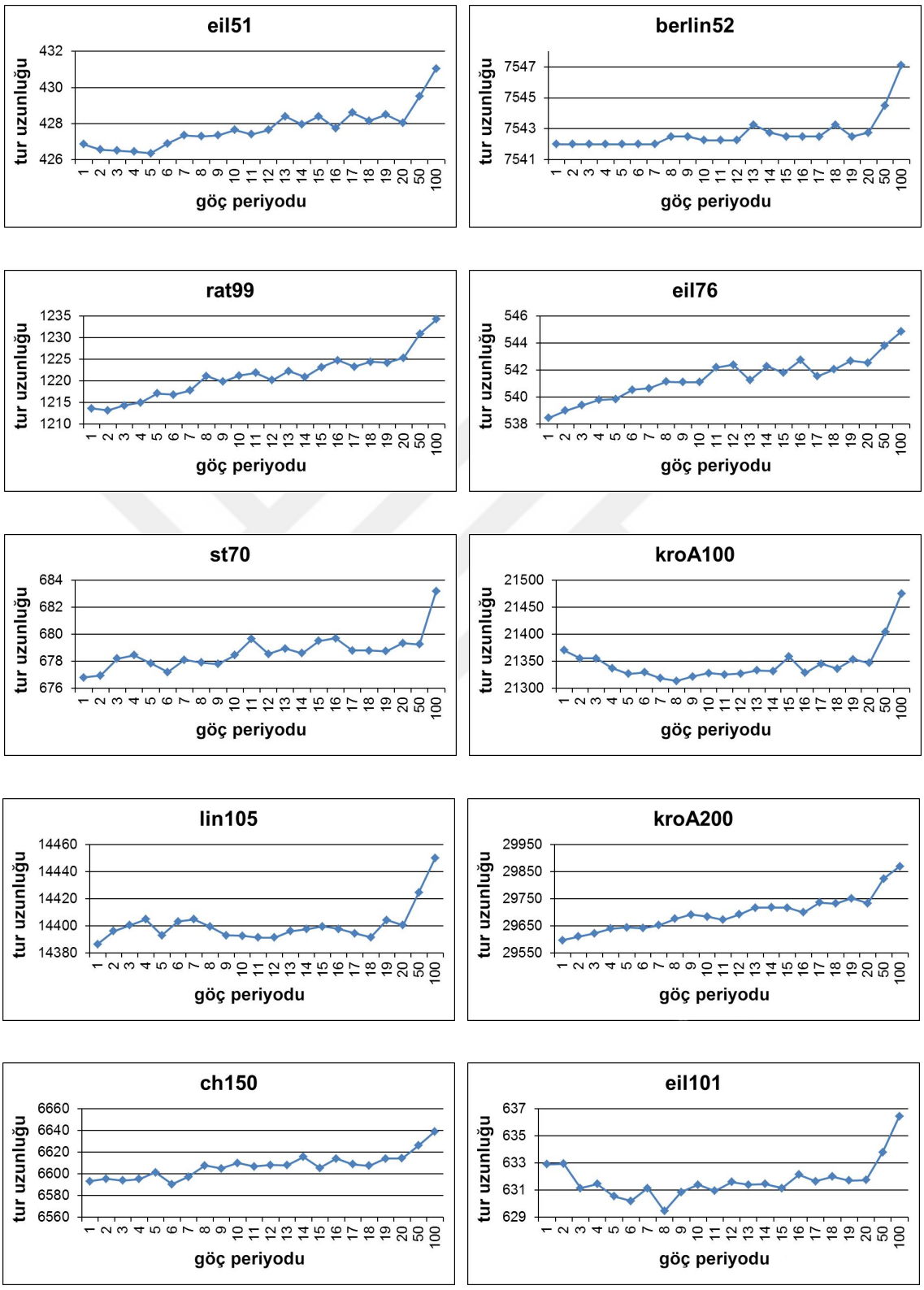
| Örnek | $\alpha$ | $\beta$ |
|---|---|---|
| eil51 | 1.11 | 1.44 |
| berlin52 | 0.95 | 1.05 |
| rat99 | 0.99 | 1.07 |
| eil76 | 0.88 | 1.50 |
| st70 | 0.94 | 1.05 |
| kroA100 | 1.01 | 1.10 |
| lin105 | 1.20 | 0.65 |
| kroA200 | 0.75 | 1.15 |
| ch150 | 0.75 | 1.20 |
| eil101 | 1.20 | 0.75 |

**Çizelge 5.24.** Farklı göç periyodları için her bir GSP örneğinin ortalama tur uzunluğu

| Göç Periyodu | eil51 | berlin52 | rat99 | eil76 | st70 | kroA100 | lin105 | kroA200 | ch150 | eil101 |
|---|---|---|---|---|---|---|---|---|---|---|
| P=1 | 426.85 | 7542.00 | 1213.65 | 538.45 | 676.80 | 21370.45 | 14386.55 | 29596.65 | 6593.05 | 632.90 |
| P=2 | 426.55 | 7542.00 | 1213.15 | 539.00 | 676.95 | 21355.50 | 14396.20 | 29611.15 | 6595.45 | 632.95 |
| P=3 | 426.50 | 7542.00 | 1214.30 | 539.40 | 678.20 | 21355.45 | 14400.50 | 29622.30 | 6593.85 | 631.15 |
| P=4 | 426.45 | 7542.00 | 1215.00 | 539.80 | 678.45 | 21337.00 | 14404.80 | 29639.75 | 6595.25 | 631.45 |
| P=5 | 426.35 | 7542.00 | 1217.10 | 539.85 | 677.85 | 21326.80 | 14393.00 | 29644.50 | 6601.40 | 630.55 |
| P=6 | 426.90 | 7542.00 | 1216.80 | 540.55 | 677.20 | 21329.55 | 14403.20 | 29642.10 | 6590.40 | 630.20 |
| P=7 | 427.35 | 7542.00 | 1217.75 | 540.65 | 678.10 | 21318.60 | 14404.80 | 29652.95 | 6597.25 | 631.15 |
| P=8 | 427.30 | 7542.50 | 1221.15 | 541.15 | 677.90 | 21313.60 | 14399.45 | 29676.40 | 6607.75 | 629.45 |
| P=9 | 427.35 | 7542.50 | 1219.85 | 541.10 | 677.80 | 21321.65 | 14393.00 | 29692.15 | 6604.95 | 630.85 |
| P=10 | 427.65 | 7542.25 | 1221.25 | 541.10 | 678.45 | 21328.25 | 14392.75 | 29684.45 | 6609.90 | 631.40 |
| P=11 | 427.40 | 7542.25 | 1221.85 | 542.20 | 679.65 | 21325.15 | 14391.40 | 29672.95 | 6606.90 | 630.95 |
| P=12 | 427.65 | 7542.25 | 1220.15 | 542.40 | 678.55 | 21327.35 | 14391.40 | 29692.65 | 6608.15 | 631.60 |
| P=13 | 428.40 | 7543.25 | 1222.25 | 541.25 | 678.95 | 21333.20 | 14396.20 | 29717.40 | 6607.95 | 631.40 |
| P=14 | 427.95 | 7542.75 | 1220.90 | 542.30 | 678.60 | 21331.75 | 14397.55 | 29718.50 | 6615.70 | 631.45 |
| P=15 | 428.40 | 7542.50 | 1223.20 | 541.80 | 679.50 | 21358.75 | 14399.65 | 29717.10 | 6605.55 | 631.15 |
| P=16 | 427.75 | 7542.50 | 1224.80 | 542.75 | 679.70 | 21328.50 | 14397.80 | 29699.55 | 6614.00 | 632.15 |
| P=17 | 428.60 | 7542.50 | 1223.30 | 541.55 | 678.80 | 21345.40 | 14394.60 | 29736.60 | 6608.85 | 631.65 |
| P=18 | 428.15 | 7543.25 | 1224.45 | 542.05 | 678.80 | 21336.50 | 14391.55 | 29732.90 | 6607.40 | 632.00 |
| P=19 | 428.50 | 7542.50 | 1224.20 | 542.70 | 678.75 | 21353.60 | 14404.35 | 29751.45 | 6614.00 | 631.70 |
| P=20 | 428.05 | 7542.75 | 1225.35 | 542.55 | 679.35 | 21346.80 | 14400.60 | 29733.90 | 6614.25 | 631.75 |
| P=50 | 429.50 | 7544.50 | 1230.90 | 543.80 | 679.25 | 21403.90 | 14424.70 | 29824.90 | 6626.20 | 633.80 |
| P=100 | 431.05 | 7547.10 | 1234.25 | 544.85 | 683.20 | 21475.60 | 14449.95 | 29870.10 | 6639.20 | 636.45 |

GSP örnekleri için göç periyodunun hesaplama süresi üzerindeki etkisi hakkında araştırmalar yapılmıştır ve sonuçlar Çizelge 5.25'de gösterilmektedir. Çeşitli göç periyodları kullanılarak, PACO-3Opt algoritması GSP örneklerini farklı hesaplama sürelerinde çözmektedir. Çizelge 5.25'de gözlendiği gibi, eğer göç periyodu uzunsa haberleşmelerin düşük frekansından dolayı hesaplama zamanı kısalmaktadır. Örneğin

eil101 için, göç periyodu sırasıyla 1'e, 2'ye ve 100'e eşitlendiği zaman hesaplama süreleri sırasıyla 105.88 saniye, 52.66 saniye ve 2.43 saniye olmaktadır.

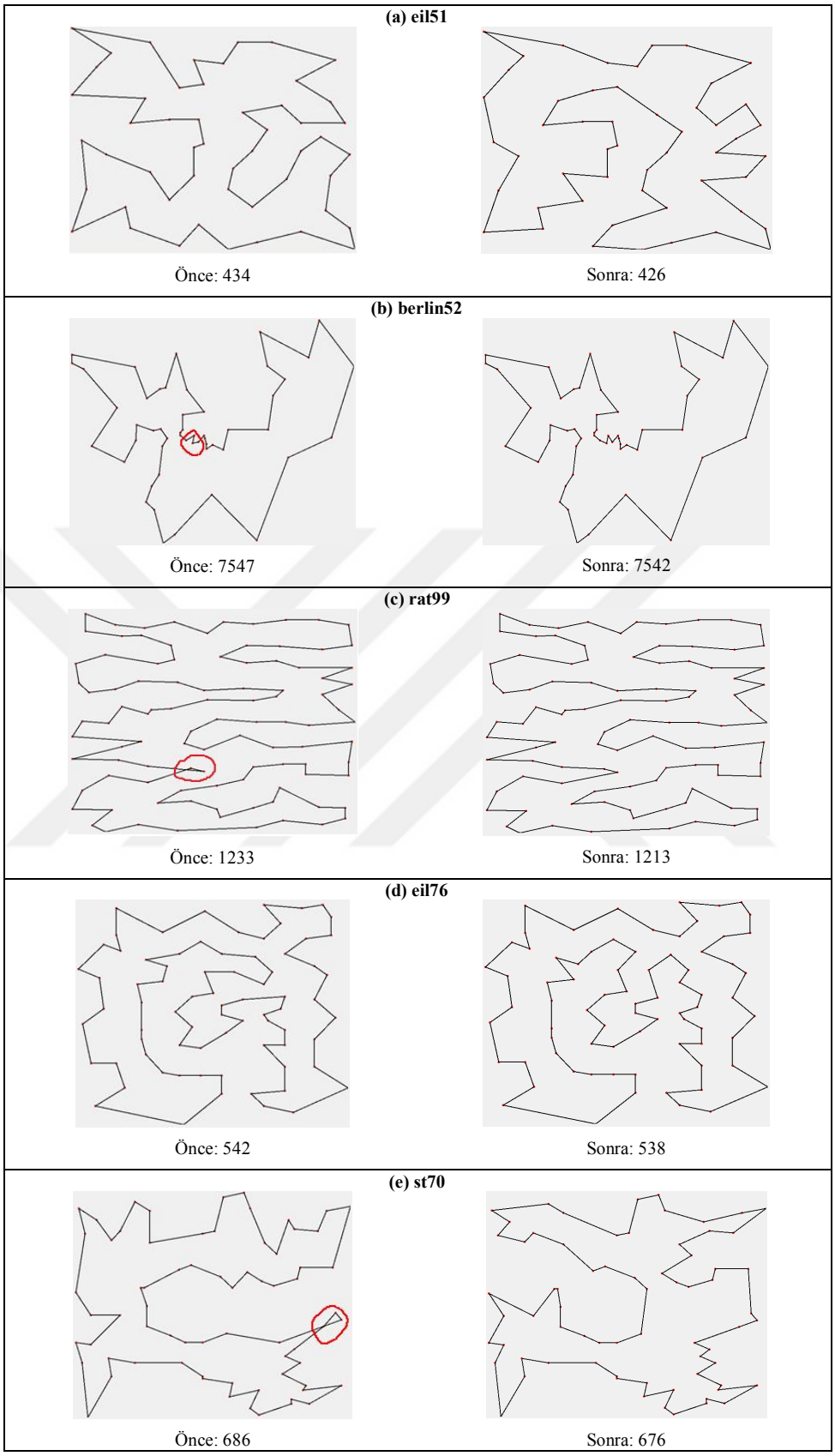



**Şekil 5.6.** GSP örnekleri için göç periyodunun tur uzunluğu üzerindeki etkisi

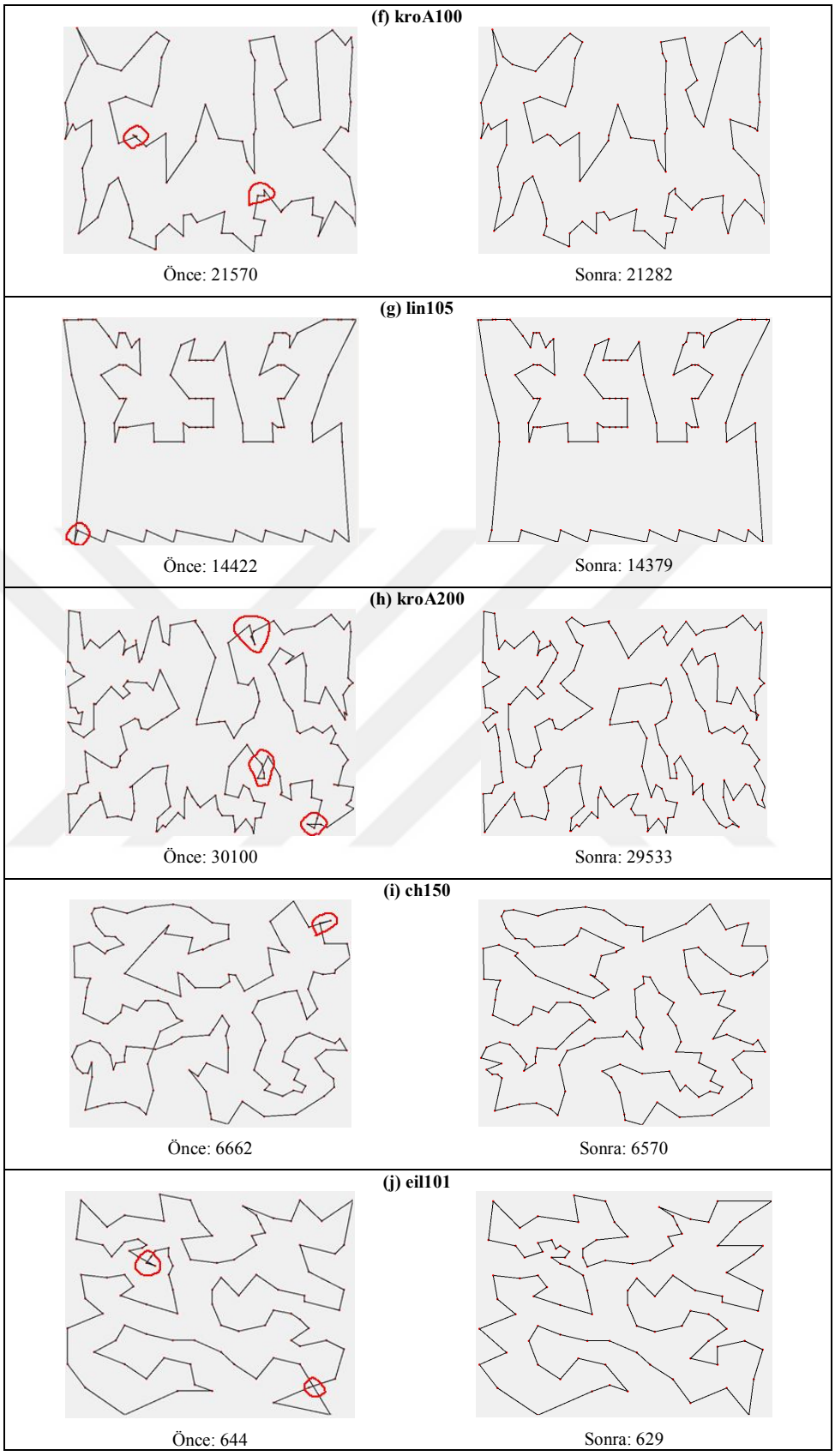
**Çizelge 5.25.** GSP örnekleri için göç periyodunun hesaplama süresi (s) üzerindeki etkisi

| Göç periyodu | eil51 | berlin52 | rat99 | eil76 | st70 | kroA100 | lin105 | kroA200 | ch150 | eil101 |
|---|---|---|---|---|---|---|---|---|---|---|
| P=1 | 10.39 | 7.81 | 88.81 | 36.53 | 31.02 | 87.52 | 54.62 | 974.68 | 386.38 | 105.88 |
| P=2 | 5.31 | 3.97 | 44.90 | 18.88 | 15.98 | 46.20 | 27.52 | 515.75 | 192.38 | 52.66 |
| P=3 | 3.56 | 2.84 | 30.56 | 12.89 | 10.82 | 32.82 | 19.40 | 353.90 | 128.73 | 34.41 |
| P=4 | 2.74 | 2.34 | 23.81 | 9.98 | 8.37 | 25.60 | 16.53 | 272.28 | 98.00 | 26.13 |
| P=5 | 2.39 | 2.10 | 19.79 | 8.18 | 6.97 | 21.10 | 14.57 | 213.12 | 79.35 | 20.79 |
| P=6 | 2.12 | 1.95 | 16.54 | 6.90 | 5.98 | 17.85 | 12.78 | 185.29 | 65.67 | 17.53 |
| P=7 | 1.87 | 1.83 | 14.48 | 6.00 | 5.10 | 15.50 | 11.97 | 160.45 | 56.77 | 15.45 |
| P=8 | 1.72 | 1.66 | 13.02 | 5.35 | 4.61 | 13.98 | 10.80 | 142.60 | 49.94 | 13.26 |
| P=9 | 1.57 | 1.55 | 11.67 | 4.86 | 4.16 | 12.59 | 10.22 | 126.56 | 43.90 | 12.17 |
| P=10 | 1.46 | 1.47 | 10.78 | 4.52 | 3.83 | 11.72 | 9.71 | 116.34 | 40.51 | 11.18 |
| P=11 | 1.37 | 1.38 | 9.83 | 4.11 | 3.51 | 10.68 | 8.80 | 105.22 | 36.76 | 10.31 |
| P=12 | 1.28 | 1.30 | 9.12 | 3.84 | 3.30 | 9.93 | 8.27 | 96.39 | 33.79 | 9.63 |
| P=13 | 1.23 | 1.25 | 8.37 | 3.63 | 3.15 | 9.22 | 7.72 | 89.35 | 31.50 | 8.93 |
| P=14 | 1.16 | 1.17 | 7.91 | 3.41 | 2.90 | 8.64 | 7.43 | 83.38 | 28.96 | 8.28 |
| P=15 | 1.13 | 1.14 | 7.58 | 3.25 | 2.80 | 8.33 | 7.23 | 79.43 | 27.46 | 8.10 |
| P=16 | 1.09 | 1.09 | 7.14 | 3.10 | 2.69 | 7.87 | 7.01 | 74.80 | 26.65 | 7.69 |
| P=17 | 1.05 | 1.05 | 6.76 | 2.96 | 2.54 | 7.47 | 6.65 | 70.08 | 24.76 | 7.26 |
| P=18 | 1.00 | 1.01 | 6.46 | 2.82 | 2.42 | 7.00 | 6.30 | 66.19 | 23.27 | 6.90 |
| P=19 | 0.98 | 1.01 | 6.31 | 2.75 | 2.38 | 6.86 | 6.10 | 64.57 | 22.44 | 6.68 |
| P=20 | 0.96 | 0.98 | 6.09 | 2.70 | 2.33 | 6.68 | 6.05 | 62.42 | 21.78 | 6.53 |
| P=50 | 0.66 | 0.67 | 3.24 | 1.56 | 1.37 | 3.51 | 3.44 | 27.75 | 10.30 | 3.44 |
| P=100 | 0.56 | 0.57 | 2.32 | 1.22 | 1.06 | 2.51 | 2.55 | 16.81 | 6.64 | 2.43 |

3-Opt algoritması iyi bilinen bir tur iyileştirme yöntemidir. Bir turdaki üst üste binmiş kenarları engellemek ve toplam tur uzunluğunu kısaltmak için kullanılır. Şekil 5.7 ve Şekil 5.8, geliştirilen yöntemde 3-Opt lokal arama algoritması kullanılmaksızın elde edilen en iyi turları -grafikte *önce* olarak- ve 3-Opt algoritması kullanılarak elde edilen en iyi turları -grafikte *sonra* olarak- göstermektedir. Üst üste binmiş kenarlar çember içinde gösterilmektedir. (a) ve (d)'de üst üste binmiş kenarlar olmamasına rağmen toplam tur uzunlukları 3-Opt algoritması kullanılması sayesinde düşürülmüştür. Şekil 5.7 ve Şekil 5.8'de görüldüğü gibi, geliştirilen yöntem lokal arama algoritması olan 3-Opt algoritmasını kullanması sayesinde üst üste binmiş kenarları gidermektedir ve bütün test örneklerinde daha iyi tur uzunlukları elde etmektedir.




**Şekil 5.7.** eil51, berlin52, rat99, eil76 ve st70 problemleri için geliştirilen yöntem tarafından 3-Opt çalıştırılmaksızın ve çalıştırdıktan sonraki bulunan en iyi turlar




**Şekil 5.8.** kroA100, lin105, kroA200, ch150 ve eil101 problemleri için geliştirilen yöntem tarafından 3-Opt çalıştırılmaksızın ve çalıştırdıktan sonraki bulunan en iyi turlar

PACO-3Opt algoritması PSO-ACO-3Opt algoritması (Mahi ve ark., 2015) ile detaylı olarak karşılaştırılmıştır ve elde edilen en iyi, en kötü ve ortalama sonuçlar Çizelge 5.26'de verilmiştir. Çizelge 5.26'deki PSO-ACO-3Opt algoritmasının sonuçları (Mahi ve ark., 2015)'den alınmıştır. İki algoritmanın karşılaştırmasındaki daha iyi sonuçlar çizelgede koyu olarak belirtilmiştir. Sonuçları bulunan en iyi tur uzunlukları açısından değerlendirecek olursak, PACO-3Opt algoritması PSO-ACO-3Opt algoritmasından daha iyi sonuçlar elde etmiştir. Dahası, PACO-3Opt algoritması eil51, berlin52, eil76, kroA100, lin105 ve eil101 örneklerinde optimum sonucu bulmaktadır. PSO-ACO-3Opt algoritması da eil51, berlin52, eil76 ve lin105 örnekleri için optimum sonucu bulmaktadır. Sonuçları ortalama tur uzunlukları açısından değerlendirecek olursak, PACO-3Opt algoritması eil76, lin105 ve ch150 hariç bütün örneklerde PSO-ACO-3Opt algoritmasından daha iyi performans göstermiştir. Algoritmaların hesaplama süreleri Çizelge 5.27'de verilmiştir. Hesaplama süresi açısından karşılaştıracak olursak, PACO-3Opt algoritması mükemmel bir performans sergilemiştir. Çizelge 5.27'deki sonuçlardan görüldüğü gibi, PACO-3Opt algoritması bütün test örneklerini PSO-ACO-3Opt algoritmasından oldukça daha hızlı çözmektedir.

Algoritmaları kıyaslamak için kullanılan bir diğer yöntem ise kutu grafiğidir. Bu yöntem ile algoritmaların başarıları ile ilgili istatistiksel bilgiler görselleştirilerek, bu bilgilerin yorumlanması kolaylaştırılır (Acılar, 2013). Kutu grafiği veri kalitesi hakkında birçok bilgiyi bir bakışta iletmek için mükemmeldir. Karşılaştırılacak verilerin kalitesini ve dağılımını belirlemek için kullanılır (Scheck, 2008). Kutu grafik; sonuçların en küçük değerini, birinci çeyrek değerini, ortanca değerini, ortalama değerini, üçüncü çeyrek değerini ve en büyük değerini sunmaktadır. Bundan dolayı, grafik karşılaştırılacak algoritmaların sonuçlarının kapsamını en küçük değerden en büyük değere kadar göstermektedir. Böylece, kutu grafiği algoritmanın istikrarı ve gürbüzlüğü hakkında bilgi vermektedir.

Şekil EK-4.1 PACO-3Opt ve PSO-ACO-3Opt algoritmalarının küçük ölçekli problemler üzerinde elde ettiği sonuçların kutu grafiğini sunmaktadır. Grafiğe göre, PACO-3Opt algoritması 3 örnek (eil76, lin105 ve ch150) dışındaki tüm örneklerde PSO-ACO-3Opt algoritmasından daha iyi sonuçlar üretmiştir. Ayrıca, birinci çeyrek ve üçüncü çeyrek arasında kalan alan açısından grafik incelendiğinde, 3 örnek (rat99, eil76 ve lin105) hariç diğer tüm örneklerde PACO-3Opt'un daha dar bir alan ürettiği gözükmektedir. Bundan dolayı, PACO-3Opt algoritmasının daha istikrarlı ve daha gürbüz olduğu söylenebilir.

**Çizelge 5.26.** Tur uzunluğu açısından PACO-3Opt algoritmasının PSO-ACO-3Opt ile karşılaştırılması

| Örnek | En iyi tur uzunluğu | | En kötü tur uzunluğu | | Ortalama tur uzunluğu | |
|---|---|---|---|---|---|---|
| | PACO-3Opt | PSO-ACO-3Opt | PACO-3Opt | PSO-ACO-3Opt | PACO-3Opt | PSO-ACO-3Opt |
| eil51 | **426** | **426** | **427** | 428 | **426.35** | 426.45 |
| berlin52 | **7542** | **7542** | **7542** | 7548 | **7542.00** | 7543.20 |
| rat99 | **1213** | 1224 | **1225** | 1230 | **1217.10** | 1227.40 |
| eil76 | **538** | **538** | 542 | **539** | 539.85 | **538.30** |
| st70 | **676** | **676** | **679** | 681 | **677.85** | 678.20 |
| kroA100 | **21282** | 21301 | **21382** | 21554 | **21326.80** | 21445.10 |
| lin105 | **14379** | **14379** | 14422 | **14381** | 14393.00 | **14379.15** |
| kroA200 | 29533 | **29468** | **29721** | 29957 | **29644.50** | 29646.05 |
| ch150 | 6570 | **6538** | 6627 | **6622** | 6601.40 | **6563.95** |
| eil101 | **629** | 631 | 639 | **638** | **630.55** | 632.70 |

**Çizelge 5.27.** Hesaplama süresi açısından PACO-3Opt algoritmasının PSO-ACO-3Opt ile karşılaştırılması

| Örnek | Süre (s) | |
|---|---|---|
| | PACO-3Opt | PSO-ACO-3Opt |
| eil51 | **2.39** | 140.50 |
| berlin52 | **2.10** | 170.46 |
| rat99 | **19.79** | 284.09 |
| eil76 | **8.18** | 220.68 |
| st70 | **6.97** | 256.89 |
| kroA100 | **21.10** | 301.32 |
| lin105 | **14.57** | 294.35 |
| kroA200 | **213.12** | 303.23 |
| ch150 | **79.35** | 286.90 |
| eil101 | **20.79** | 302.15 |

PACO-3Opt algoritmasının performansını doğrulamak için, Çizelge 5.28'da verilen 14 algoritma PACO-3Opt algoritması ile karşılaştırılmak için seçilmiştir. Bu 14 algoritmanın karakterleri ve özellikleri Kaynak Araştırması bölümünde özet olarak açıklanmıştır.

**Çizelge 5.28.** Kıyaslama için kullanılan algoritmalar

| Kısa adı | Uzun adı | Referansı |
|---|---|---|
| PSO-ACO-3Opt | Parçacık sürü optimizasyonu, karınca koloni algoritması ve 3-Opt algoritmasına dayalı hibrit bir yöntem | (Mahi ve ark., 2015) |
| ACO+ABC | Karınca koloni algoritması ve yapay arı koloni algoritmasına dayalı hibrit bir yöntem | (Gündüz ve ark., 2014) |
| ACO+Taguchi yöntemi | Taguchi yöntemi ile birleştirilmiş karınca koloni algoritması | (Peker ve ark., 2013) |
| WFA+2-Opt | 2-Opt lokal arama ile birleştirilmiş su akışı algoritması | (Othman ve ark., 2013) |
| WFA+3-Opt | 3-Opt lokal arama ile birleştirilmiş su akışı algoritması | (Othman ve ark., 2013) |
| IVRS+2Opt | 2-Opt lokal arama ile birleştirilmiş bireysel varyasyon ve rotalama stratejisi | (Jun-man ve Yi, 2012) |
| ACO+2opt | 2-Opt lokal arama ile birleştirilmiş karınca koloni algoritması | (Jun-man ve Yi, 2012) |
| HACO | Çapraz-silme yöntemi ile birleştirilmiş karınca koloni algoritması | (Junqiang ve Aijia, 2012) |
| CGAS | Kooperatif genetik karınca sistemi | (Dong ve ark., 2012) |
| SA ACO PSO | Benzetimli tavlama algoritmasına, genetik algoritmaya, karınca koloni sistemine ve parçacık sürü optimizasyonuna dayalı bir işbirlikçi yöntem | (Chen ve Chien, 2011) |
| İyileştirilmiş RABNET-TSP | Sinir ağları ve bağışıklık sistemine dayalı iyileştirilmiş sezgisel algoritma | (Masutti ve de Castro, 2009) |
| RABNET-TSP | Sinir ağları ve bağışıklık sistemine dayalı sezgisel algoritma | (Pasti ve De Castro, 2006) |
| ACOMAC | Çoklu-karınca klan konsepti | (Tsai ve ark., 2004) |
| ACOMAC+NN | Çoklu en yakın komşu ile birleştirilmiş çoklu-karınca klan konsepti | (Tsai ve ark., 2004) |

Çizelge 5.29 her bir GSP örneği için PACO-3Opt algoritmasının bu 14 algoritma ile karşılaştırmasını sunmaktadır. Çizelge 5.29'de, en iyi sonuçlar koyu olarak yazılmıştır ve '-' simgesi ise orijinal kaynakta ilgili GSP örneği için herhangi bir sonucun olmadığı manasına gelmektedir. Çözümlerin kalitesi ortalama tur uzunluğu, standart sapma ve optimumdan rölatif sapma $\omega$ açısından verilmektedir. Rölatif sapma $\omega$ formülü Denklem (5.16)'da gösterilmektedir (Marinakis ve ark., 2011). $c_{avg}$ ilgili algoritma tarafından bulunan tur uzunluklarının ortalama maliyetini göstermektedir ve $c_{opt}$ optimal çözümün maliyetidir.

$$\omega(\%) = \frac{c_{avg} - c_{opt}}{c_{opt}} \times 100 \tag{5.16}$$

Rölatif sapmaya ve ortalama tur uzunluğuna göre, PACO-3Opt algoritması eil51, berlin52, rat99, st70, kroA200 ve eil101 örneklerinde 14 algoritmadan daha iyi sonuçlar üretmektedir. PSO-ACO-3Opt algoritması eil76 örneğinde diğer algoritmalardan daha iyi sonuç bulmaktadır. PACO-3Opt algoritması da PSO-ACO-3Opt algoritması gibi eil76 örneğinde optimuma yakın bir sonuç bulmaktadır. WFA-2Opt algoritması kroA100 ve lin105 örneklerinde diğer algoritmalardan daha iyi sonuçlar elde etmektedir. PACO-3Opt'un rölatif sapması bütün GSP örnekleri için 1.2%'den daha azdır. Ayrıca, PACO-3Opt'un sonuçlarının standart sapmaları düşüktür. Bunun anlamı PACO-3Opt algoritması farklı problemler için diğer algoritmalardan daha kararlı sonuçlar elde etmektedir.

Daha önce bahsedildiği gibi, tek koloniye sahip ACO algoritmasının en büyük eksikliği aramanın çok erken durgunluğudur. Bu eksikliğin üstesinden gelmek için, çoklu-koloniye sahip PACO-3Opt algoritmasında bulunan global en iyi tur koloniler arasında ara ara paylaşılmaktadır. Böylece, PACO-3Opt aramayı daha kaliteli çözümlere doğru yönlendirmektedir. Şekil 5.9 kıyaslama örneklerinde bulunan en iyi tur uzunluklarının eğri gelişim diyagramlarını göstermektedir. Bu diyagramlardan gözlendiği gibi, PACO-3Opt en iyi çözüme doğru hızlıca yakınsamaktadır. Örneğin PACO-3Opt tarafından en iyi turları bulunan eil51, berlin52 ve kroA100 problemlerinin yakınsama iterasyonu yaklaşık olarak 200'dür. Bu arada, eil76'da görüldüğü gibi PACO-3Opt aramayı ısrarla sürdürmektedir ve en iyi sonucu bulmaktadır.

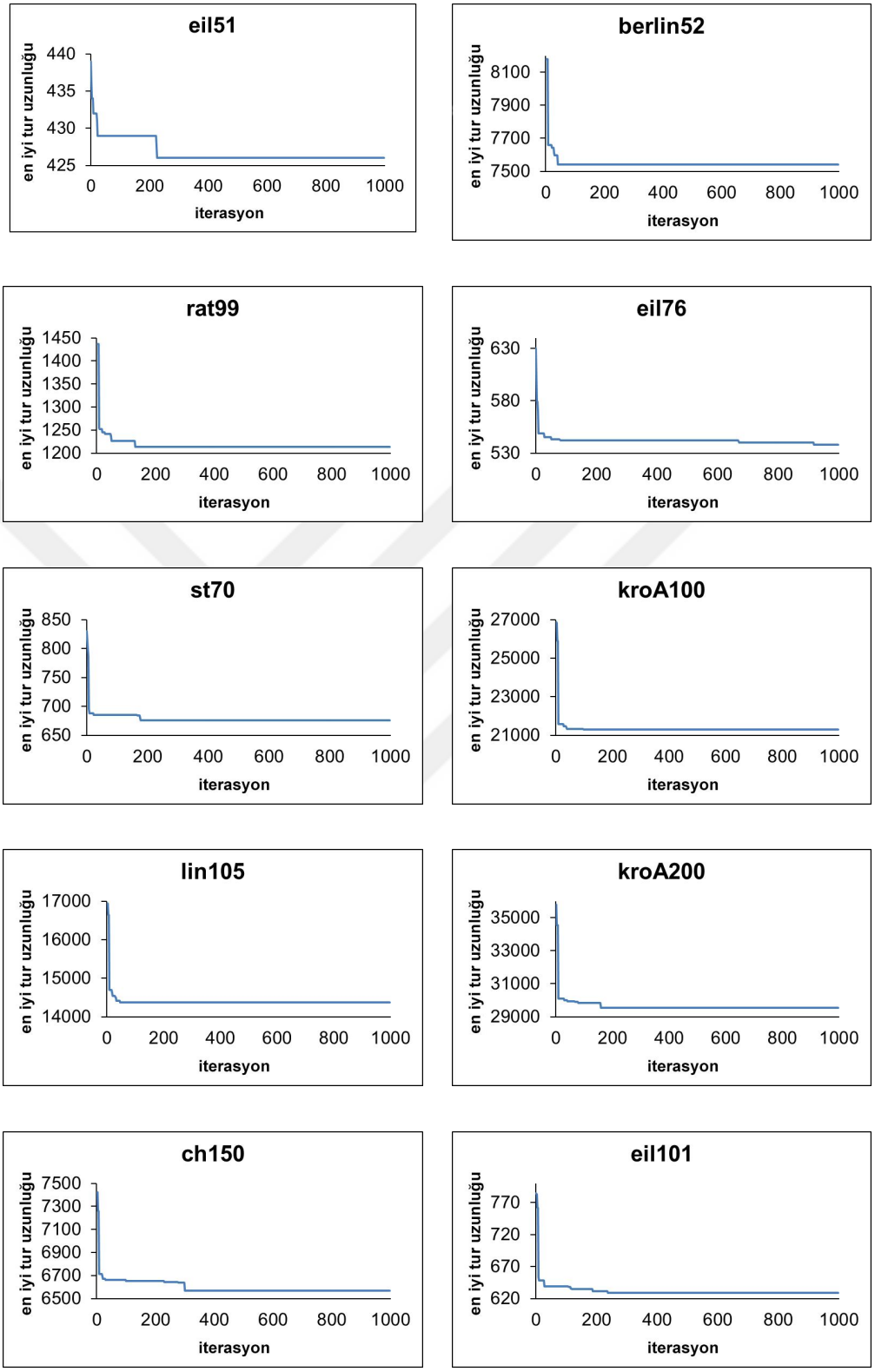
Göç sürecinde çözüm kalitesi ve hesaplama zamanı arasında her zaman bir ödün verme vardır. Simülasyon sonuçlarına göre, kısa aralıklı göçlerde GSP örnekleri için daha iyi sonuçlar elde edilecektir, fakat hesaplama zamanı artacaktır. Çizelge 5.24 ve Şekil 5.6'da görüldüğü gibi göç periyodu $P$ 5 civarında iken daha iyi sonuçlar elde edilmektedir. Bundan dolayı, GSP örneklerini çözmek için her 5 iterasyonda bir göç sürecini gerçekleştirmek önerilmektedir.

**Çizelge 5.29.** Küçük-ölçekli GSP problemlerinde PACO-3Opt algoritmasının diğer algoritmalar ile karşılaştırılması

| Yöntem | Örnek<br>BES | eil51<br>426 | berlin52<br>7542 | rat99<br>1211 | eil76<br>538 | st70<br>675 |
|---|---|---|---|---|---|---|
| PACO-3Opt | Ort. | **426.35** | **7542.00** | **1217.10** | 539.85 | **677.85** |
| | SS | 0.49 | 0.00 | 4.01 | 1.09 | 0.99 |
| | $\omega$(%) | 0.08 | 0.00 | 0.50 | 0.34 | 0.42 |
| PSO-ACO-3Opt (2015) | Ort. | 426.45 | 7543.20 | 1227.40 | **538.30** | 678.20 |
| | SS | 0.61 | 2.37 | 1.98 | 0.47 | 1.47 |
| | $\omega$(%) | 0.11 | 0.02 | 1.35 | 0.06 | 0.47 |
| ACOMAC (2004) | Ort. | 430.68 | - | - | 555.70 | - |
| | SS | - | - | - | - | - |
| | $\omega$(%) | 1.10 | - | - | 3.29 | - |
| ACOMAC+NN (2004) | Ort. | 430.68 | - | - | 555.90 | - |
| | SS | - | - | - | - | - |
| | $\omega$(%) | 1.10 | - | - | 3.33 | - |
| RABNET-TSP (2006) | Ort. | 438.70 | 8073.97 | - | 556.10 | - |
| | SS | 3.52 | 270.14 | - | 8.03 | - |
| | $\omega$(%) | 2.98 | 7.05 | - | 3.36 | - |
| İyileştirilmiş RABNET-TSP (2009) | Ort. | 437.47 | 7932.50 | - | 556.33 | - |
| | SS | 4.20 | 277.25 | - | 5.30 | - |
| | $\omega$(%) | 2.69 | 5.18 | - | 3.41 | - |
| SA ACO PSO (2011) | Ort. | 427.27 | **7542.00** | - | 540.20 | - |
| | SS | 0.45 | 0.00 | - | 2.94 | - |
| | $\omega$(%) | 0.30 | 0.00 | - | 0.41 | - |
| IVRS+2opt (2012) | Ort. | 431.10 | 7547.23 | - | - | - |
| | SS | - | - | - | - | - |
| | $\omega$(%) | 1.20 | 0.07 | - | - | - |
| ACO+2opt (2012) | Ort. | 439.25 | 7556.58 | - | - | - |
| | SS | - | - | - | - | - |
| | $\omega$(%) | 3.11 | 0.19 | - | - | - |
| HACO (2012) | Ort. | 431.20 | 7560.54 | 1241.33 | - | - |
| | SS | 2.00 | 67.48 | 9.60 | - | - |
| | $\omega$(%) | 1.22 | 0.23 | 3.69 | - | - |
| CGAS (2012) | Ort. | - | 7634.00 | - | 542.00 | - |
| | SS | - | - | - | - | - |
| | $\omega$(%) | - | 1.22 | - | 0.74 | - |
| WFA + 2-Opt (2013) | Ort. | 426.65 | **7542.00** | - | 541.22 | - |
| | SS | 0.66 | 0.00 | - | 0.66 | - |
| | $\omega$(%) | 0.15 | 0.00 | - | 0.60 | - |
| WFA + 3-Opt (2013) | Ort. | 426.60 | **7542.00** | - | 539.44 | - |
| | SS | 0.52 | 0.00 | - | 1.51 | - |
| | $\omega$(%) | 0.14 | 0.00 | - | 0.27 | - |
| ACO + Taguchi yöntemi (2013) | Ort. | 435.40 | 7635.40 | | 565.50 | |
| | SS | - | - | - | - | - |
| | $\omega$(%) | 2.21 | 1.24 | - | 5.11 | - |
| ACO + ABC (2014) | Ort. | 443.39 | 7544.37 | - | 557.98 | 700.58 |
| | SS | 5.25 | 0.00 | - | 4.10 | 7.51 |
| | $\omega$(%) | 4.08 | 0.03 | - | 3.71 | 3.79 |

| Yöntem | Örnek<br>BES | kroA100<br>21282 | lin105<br>14379 | kroA200<br>29368 | ch150<br>6528 | eil101<br>629 |
|---|---|---|---|---|---|---|
| PACO-3Opt | Ort. | 21326.80 | 14393.00 | **29644.50** | 6601.40 | **630.55** |
| | SS | 33.72 | 19.76 | 53.43 | 15.01 | 2.63 |
| | $\omega$(%) | 0.21 | 0.10 | 0.94 | 1.12 | 0.25 |
| PSO-ACO-3Opt (2015) | Ort. | 21445.10 | 14379.15 | 29646.05 | 6563.95 | 632.70 |
| | SS | 78.24 | 0.48 | 114.71 | 27.58 | 2.12 |
| | $\omega$(%) | 0.77 | 0.00 | 0.95 | 0.55 | 0.59 |
| ACOMAC (2004) | Ort. | 21457.00 | - | - | - | - |
| | SS | - | - | - | - | - |
| | $\omega$(%) | 0.82 | - | - | - | - |
| ACOMAC+NN (2004) | Ort. | 21433.30 | - | - | - | - |
| | SS | - | - | - | - | - |
| | $\omega$(%) | 0.71 | - | - | - | - |
| RABNET-TSP (2006) | Ort. | 21868.47 | 14702.17 | 30257.53 | 6753.20 | 654.83 |
| | SS | 245.76 | 328.37 | 342.98 | 83.01 | 6.57 |
| | $\omega$(%) | 2.76 | 2.25 | 3.03 | 3.45 | 4.11 |
| İyileştirilmiş RABNET-TSP (2009) | Ort. | 21522.73 | 14400.7 | 30190.27 | 6738.37 | 648.63 |
| | SS | 93.34 | 44.03 | 273.38 | 76.14 | 3.85 |
| | $\omega$(%) | 1.13 | 0.15 | 2.80 | 3.22 | 3.12 |
| SA ACO PSO (2011) | Ort. | 21370.30 | 14406.37 | 29738.73 | **6563.70** | 635.23 |
| | SS | 123.36 | 37.28 | 356.07 | 22.45 | 3.59 |
| | $\omega$(%) | 0.41 | 0.19 | 1.26 | 0.55 | 0.99 |
| IVRS+2opt (2012) | Ort. | 21498.61 | - | - | - | 648.67 |
| | SS | - | - | - | - | - |
| | $\omega$(%) | 1.02 | - | - | - | 3.13 |
| ACO+2opt (2012) | Ort. | 23441.80 | - | - | - | 672.37 |
| | SS | - | - | - | - | - |
| | $\omega$(%) | 10.15 | - | - | - | 6.90 |
| HACO (2012) | Ort. | - | - | - | - | - |
| | SS | - | - | - | - | - |
| | $\omega$(%) | - | - | - | - | - |
| CGAS (2012) | Ort. | 21437.00 | - | 29946.00 | - | - |
| | SS | - | - | - | - | - |
| | $\omega$(%) | 0.73 | - | 1.97 | - | - |
| WFA + 2-Opt (2013) | Ort. | **21282.00** | **14379.00** | 29654.03 | 6572.13 | 639.87 |
| | SS | 0.00 | 0.00 | 151.42 | 13.84 | 2.88 |
| | $\omega$(%) | 0.00 | 0.00 | 0.97 | 0.68 | 1.73 |
| WFA + 3-Opt (2013) | Ort. | 21282.80 | 14459.40 | 29646.50 | 6700.10 | 633.50 |
| | SS | 0.00 | 1.38 | 110.91 | 60.82 | 3.47 |
| | $\omega$(%) | 0.00 | 0.56 | 0.95 | 2.64 | 0.72 |
| ACO + Taguchi yöntemi (2013) | Ort. | 21567.10 | 14475.20 | | | 655.00 |
| | SS | - | - | - | - | - |
| | $\omega$(%) | 1.34 | 0.67 | - | - | 4.13 |
| ACO + ABC (2014) | Ort. | 22435.31 | - | - | 6677.12 | 683.39 |
| | SS | 231.34 | - | - | 19.30 | 6.56 |
| | $\omega$(%) | 5.42 | - | - | 2.28 | 8.65 |

BES: bilinen en iyi sonuç, Ort.: ortalama tur uzunluğu, SS: standart sapma, $\omega$(%): rölatif sapma




**Şekil 5.9.** GSP örneklerinde bulunan en iyi tur uzunluklarının eğri gelişim diyagramı

### 5.2.2. Büyük-ölçekli problemler üzerinde PACO-3Opt algoritmasının değerlendirilmesi

Bu bölümde, problem boyutları 400 ile 800 arasında olan 11 adet büyük-ölçekli GSP örneği kullanılmaktadır: rd400, fl417, pr439, pcb442, d493, u574, rat575, p654, d657, u724 ve rat783. Bunların bilinen en iyi sonuçları Çizelge 5.32'da gösterilmektedir. Bu bölümdeki tüm deneylerde, PACO-3Opt algoritmasının parametreleri Çizelge 5.30'deki gibi belirlenmiştir. Çizelge 5.30'den görüldüğü gibi, PACO-3Opt algoritması hem iki bilgisayara ($n$=2) sahip bir küme üzerinde hem de dört bilgisayara ($n$=4) sahip bir küme üzerinde çalıştırılmaktadır. Her bir kümede bir tane sunucu koloni bulunmaktadır. Kümedeki diğer bilgisayarlar ise istemci koloniler içindir. PACO-3Opt algoritmasının performansını adilce karşılaştırmak için, PACO-3Opt algoritmasının parametreleri Yong (2015)'un yayınındaki ile aynıdır. Popülasyon boyutu 20'dir. Feromon buharlaşma oranı $\rho$ 0.1, $Q$ sabiti 30 ve maksimum iterasyon sayısı 1000 olarak ayarlanmıştır. Yong (2015) büyük-ölçekli GSP örnekleri için $\alpha$ ve $\beta$ parametre değerlerinin sırasıyla 3 ve 2'ye eşit olduğu zaman daha iyi sonuçların elde edildiğini belirtmektedir. Bundan dolayı $\alpha$ ve $\beta$ parametrelerin değerleri sırasıyla 3 ve 2'dir. Küçük-ölçekli problemlerde, $P$ göç periyodu 5 olduğu zaman daha iyi sonuçlar elde edilmektedir. Bundan dolayı, büyük-ölçekli problemlerde de göç periyodu 5 olarak ayarlanmıştır. Ayrıca, her bir test bağımsız olarak 20 kez tekrarlanmıştır.

**Çizelge 5.30.** Büyük-ölçekli GSP örnekleri için parametre değerleri

| $n$ | $m$ | $P$ | $\alpha$ | $\beta$ | $\rho$ | $Q$ | maksimum iterasyon |
|---|---|---|---|---|---|---|---|
| 2 | 10 | 5 | 3 | 2 | 0.1 | 30 | 1000 |
| 4 | 5 | 5 | 3 | 2 | 0.1 | 30 | 1000 |

PACO-3Opt algoritması PSO-ACO-3Opt algoritması ile karşılaştırılmıştır (Mahi ve ark., 2015). En iyi, ortalama ve en kötü tur uzunluğu sonuçları sırasıyla Çizelge 5.31, Çizelge 5.32 ve Çizelge 5.33'de ve hesaplama süreleri Çizelge 5.34'de verilmiştir. Sonuçları en iyi tur uzunluğu açısından değerlendirecek olursak, 4 kolonili PACO-3Opt algoritması PSO-ACO-3Opt algoritmasından ve 2 kolonili PACO-3Opt'dan daha iyi sonuçlar elde etmiştir. Ortalama tur uzunlukları açısından değerlendirecek olursak, 2 kolonili PACO-3Opt ve 4 kolonili PACO-3Opt benzer performansa sahiptir ve bunların sonuçları PSO-ACO-3Opt'dan açıkçası daha iyidir. Bu iki algoritma fl417 hariç bütün problemlerde PSO-ACO-3Opt'dan daha iyi performans göstermektedirler. Sonuçları en

kötü tur uzunluğu açısından değerlendirecek olursak, 2 kolonili ve 4 kolonili PACO-3Opt algoritmaları PSO-ACO-3Opt'dan daha iyi sonuçlar elde etmişlerdir. Hesaplama zamanı açısından kıyaslayacak olursak, PACO-3Opt paralel çalışma yeteneğinden dolayı mükemmel bir performans sergilemiştir. Çizelge 5.34'daki sonuçlardan görüldüğü gibi, PACO-3Opt bütün test örneklerini PSO-ACO-3Opt'dan oldukça daha hızlı çözmektedir. Ayrıca sonuçlardan görüldüğü gibi, 4 kolonili PACO-3Opt'un hızı 2 kolonili PACO-3Opt'in hızının nerdeyse iki katıdır.

**Çizelge 5.31.** En iyi tur uzunluğu açısından PACO-3Opt algoritmasının PSO-ACO-3Opt ile karşılaştırılması

| Örnek | BES | En iyi tur uzunluğu | | |
|---|---|---|---|---|
| | | PACO-3Opt n=2 | PACO-3Opt n=4 | PSO-ACO-3Opt |
| rd400 | 15281 | **15527** | 15578 | 15594 |
| fl417 | 11861 | 11958 | 11972 | **11947** |
| pr439 | 107217 | 108516 | **108482** | 108530 |
| pcb442 | 50778 | 52057 | **51962** | 52131 |
| d493 | 35002 | 35742 | **35735** | 35789 |
| u574 | 36905 | 37855 | 37981 | **37818** |
| rat575 | 6773 | 7009 | 7003 | **6987** |
| p654 | 34643 | **35027** | 35045 | 35052 |
| d657 | 48912 | 50230 | **50206** | 50291 |
| u724 | 41910 | 42893 | **42764** | 43172 |
| rat783 | 8806 | 9112 | **9111** | 9128 |

BES: bilinen en iyi sonuç

**Çizelge 5.32.** Ortalama tur uzunluğu açısından PACO-3Opt algoritmasının PSO-ACO-3Opt ile karşılaştırılması

| Örnek | BES | Ortalama tur uzunluğu | | |
|---|---|---|---|---|
| | | PACO-3Opt n=2 | PACO-3Opt n=4 | PSO-ACO-3Opt |
| rd400 | 15281 | **15601.0** | 15613.9 | 15691.3 |
| fl417 | 11861 | 11991.9 | 11987.4 | **11980.4** |
| pr439 | 107217 | 108736.6 | **108702.0** | 108965.4 |
| pcb442 | 50778 | 52267.5 | **52202.4** | 52368.1 |
| d493 | 35002 | **35811.0** | 35841.0 | 35973.8 |
| u574 | 36905 | **38020.3** | 38030.7 | 38112.9 |
| rat575 | 6773 | 7024.5 | **7012.4** | 7018.6 |
| p654 | 34643 | **35065.8** | 35075.0 | 35098.2 |
| d657 | 48912 | 50352.9 | **50277.5** | 50475.5 |
| u724 | 41910 | **43100.1** | 43122.5 | 43300.3 |
| rat783 | 8806 | 9130.7 | **9127.3** | 9138.1 |

BES: bilinen en iyi sonuç

**Çizelge 5.33.** En kötü tur uzunluğu açısından PACO-3Opt algoritmanın PSO-ACO-3Opt ile karşılaştırılması

| Örnek | BES | En kötü tur uzunluğu | | |
|---|---|---|---|---|
| | | PACO-3Opt n=2 | PACO-3Opt n=4 | PSO-ACO-3Opt |
| rd400 | 15281 | **15663** | 15667 | 15757 |
| fl417 | 11861 | 12005 | **12000** | 12003 |
| pr439 | 107217 | **108889** | 108973 | 109341 |
| pcb442 | 50778 | 52393 | **52283** | 52629 |
| d493 | 35002 | **35927** | 35982 | 36045 |
| u574 | 36905 | **38157** | 38165 | 38232 |
| rat575 | 6773 | 7041 | **7037** | 7049 |
| p654 | 34643 | 35123 | **35116** | 35145 |
| d657 | 48912 | 50480 | **50386** | 50622 |
| u724 | 41910 | **43240** | 43272 | 43472 |
| rat783 | 8806 | **9145** | 9152 | 9149 |

BES: bilinen en iyi sonuç

**Çizelge 5.34.** Büyük-ölçekli problemler üzerinde algoritmaların hesaplama süreleri (s)

| Örnek | PACO-3Opt n=2 | PACO-3Opt n=4 | PSO-ACO-3Opt |
|---|---|---|---|
| rd400 | 2841 | **1496** | 7191 |
| fl417 | 4027 | **2046** | 10019 |
| pr439 | 4231 | **2132** | 11369 |
| pcb442 | 3879 | **2088** | 10572 |
| d493 | 6248 | **3175** | 16672 |
| u574 | 10654 | **5460** | 27884 |
| rat575 | 9751 | **5004** | 24648 |
| p654 | 17681 | **9105** | 45732 |
| d657 | 17088 | **8715** | 44225 |
| u724 | 23282 | **11458** | 55229 |
| rat783 | 28657 | **14277** | 75329 |

Şekil EK-4.2 PACO-3Opt ($n$=4) ve PSO-ACO-3Opt algoritmalarının büyük ölçekli problemler üzerinde elde ettiği sonuçların kutu grafiğini sunmaktadır. Grafiğe göre, PACO-3Opt algoritması 11 örneğin 10'unda PSO-ACO-3Opt algoritmasından daha iyi sonuçlar elde etmiştir. PSO-ACO-3Opt algoritması sadece fl417 örneğinde başarılıdır. Ayrıca, en küçük değer ile en büyük değer arasında kalan alan açısından grafik incelendiğinde, 2 örnek (u724 ve rat783) dışındaki diğer tüm örneklerde PACO-3Opt'un daha dar bir alan ürettiği gözükmektedir. Bundan dolayı, PACO-3Opt algoritmasının daha istikrarlı ve daha gürbüz olduğu söylenebilir.

Büyük-ölçekli problemler üzerinde PACO-3Opt algoritmasının performansını doğrulamak için Çizelge 5.35'de verilen 8 algoritma PACO-3Opt algoritması ile karşılaştırılmak için seçilmiştir. Bu 8 algoritmanın karakterleri ve özellikleri Kaynak Araştırması bölümünde özet olarak açıklanmıştır.

**Çizelge 5.35.** Kıyaslama için kullanılan algoritmalar

| Kısa adı | Uzun adı | Referansı |
|---|---|---|
| PSO-ACO-3Opt | Parçacık sürü optimizasyonu, karınca koloni algoritması ve 3-Opt algoritmasına dayalı hibrit bir yöntem | (Mahi ve ark., 2015) |
| HMMA | Lokal arama algoritmalı hibrit maksimum-minimum karınca sistemi | (Yong, 2015) |
| HHNN | En yakın komşu algoritması ve en kısa rota ve ekleme algoritmasına dayalı hiper-sezgisel algoritma | (Kızılateş ve ark., 2015) |
| PMSOM | Kendi-organizeli haritaya dayalı paralelleştirilmiş böl-ve-fethet yöntemi | (Avşar ve Aliabadi, 2015) |
| DIWO | Ayrık istilacı yabani ot optimizasyonunu | (Zhou ve ark., 2015) |
| HPSACO | Karınca koloni algoritması ve parçacık sürü optimizasyonu tabanlı hibrit sezgisel algoritma | (Jia, 2015) |
| HCACO | Karınca koloni algoritması ve kümeleme algoritmaları tabanlı hiyerarşik algoritma | (Jiang ve ark., 2014) |
| HGA | Lokal optimizasyon stratejiye sahip hibrit genetik algoritma | (Wang, 2014) |

Çizelge 5.36 her bir GSP örneği için ortalama tur uzunluklarını ve en iyi tur uzunluklarını sunmaktadır. Çizelge 5.36'deki '-' karakteri orijinal kaynakta ilgili problemin herhangi bir sonucunun olmadığı anlamına gelmektedir. Ortalama tur uzunluğuna göre, PACO-3Opt algoritması fl417 ve pcb442 hariç bütün örneklerde diğer algoritmalardan daha iyi performans sergilemektedir. PSO-ACO-3Opt algoritması fl417 örneğinde diğer algoritmalardan daha iyi sonuca ulaşmaktadır. DIWO algoritması pcb442 örneğinde diğer algoritmalardan daha iyi sonucu bulmaktadır. En iyi tur uzunluğu açısından değerlendirecek olursak, PACO-3Opt pr439, d493, p654, d657, u724 ve rat783 örneklerinde daha iyi sonuçlara ulaşmaktadır.

Sonuç olarak, deneysel sonuçlar göstermiştir ki PACO-3Opt algoritması küçük-ölçekli ve büyük-ölçekli GSP örneklerinin çözümünde iyi bir performans elde etmiştir. PACO-3Opt algoritması diğer optimizasyon algoritmaları ile oldukça rekabetçidir ve GSP örneklerini çözen herhangi bir meta sezgisel algoritmaya alternatif olabilmektedir.

**Çizelge 5.36.** Büyük-ölçekli GSP örnekleri için öneriler algoritmanın diğer algoritmalar ile karşılaştırılması

| Örnek | | PACO-3Opt n=4 (2016) | PSO-ACO-3Opt (2015) | HMMA (2015) | HHNN (2015) | PMSOM (2015) | DIWO (2015) | HPSACO (2015) | HCACO (2014) | HGA (2014) |
|---|---|---|---|---|---|---|---|---|---|---|
| rd400 | Ort. | **15613.9** | 15691.3 | 16723.68 | 17728.00 | - | 15651.24 | 16513.00 | - | 15852.74 |
| | Eniyi | 15578 | 15594 | 16534 | - | - | **15543** | 16067 | - | - |
| fl417 | Ort. | 11987.4 | **11980.4** | 12897.84 | - | - | - | - | - | - |
| | Eniyi | 11972 | **11947** | 12543 | - | - | - | - | - | - |
| pr439 | Ort. | **108702.0** | 108965.4 | 116885.91 | - | - | - | - | - | 109249.66 |
| | Eniyi | **108482** | 108530 | 114095 | - | - | - | - | - | - |
| pcb442 | Ort. | 52202.4 | 52368.1 | 55670.17 | - | 53362.6 | **51881.46** | - | 55326.00 | 52376.26 |
| | Eniyi | 51962 | 52131 | 54401 | - | 52631 | **51597** | - | 54838 | - |
| d493 | Ort. | **35841.0** | 35973.8 | 38410.25 | 38988.00 | - | - | - | - | - |
| | Eniyi | **35735** | 35789 | 37187 | - | - | - | - | - | - |
| u574 | Ort. | **38030.7** | 38112.9 | - | | - | - | - | 39411.00 | - |
| | Eniyi | 37981 | **37818** | - | | - | - | - | 39228 | - |
| rat575 | Ort. | **7012.4** | 7018.6 | 7745.63 | 7763.00 | - | - | - | - | - |
| | Eniyi | 7003 | **6987** | 7575 | - | - | - | - | - | - |
| p654 | Ort. | **35075.0** | 35098.2 | 37931.4 | | 36399.4 | - | - | - | - |
| | Eniyi | **35045** | 35052 | 37044 | | 35953 | - | - | - | - |
| d657 | Ort. | **50277.5** | 50475.5 | 55554.31 | 56134.00 | - | - | - | 53234.00 | - |
| | Eniyi | **50206** | 50291 | 55163 | - | - | - | - | 52665 | - |
| u724 | Ort. | **43122.5** | 43300.3 | 47132.71 | - | 43980.35 | - | - | - | - |
| | Eniyi | **42764** | 43172 | 46662 | - | 43704 | - | - | - | - |
| rat783 | Ort. | **9127.3** | 9138.1 | 10256.23 | 10189.00 | - | - | - | 9505.00 | - |
| | Eniyi | **9111** | 9128 | 10149 | - | - | - | - | 9453 | - |

*Ort.*: ortalama tur uzunluğu, *Eniyi*: en iyi tur uzunluğu

### 5.2.3. PACO-3Opt algoritmasının seri versiyonu ile karşılaştırılması

Bu bölümde, PACO-3Opt algoritması seri versiyonu ile karşılaştırılmaktadır. Algoritmanın seri versiyonu, PACO-3Opt algoritmasının tek bir işlemci üzerinde çalışmasıyla elde edilmektedir. Küçük ölçekli problemlerde PACO-3Opt algoritması ve seri versiyonu iki koloniye sahiptirler. Büyük ölçekli problemlerde ise PACO-3Opt algoritmasını hem iki bilgisayara ($n$=2) sahip bir küme üzerinde hem de dört bilgisayara ($n$=4) sahip bir küme üzerinde çalıştırmaktayız. Bundan dolayı büyük ölçekli problemlerde algoritmanın seri versiyonunu da hem iki kolonili (Seri $n$=2) olarak hem de dört kolonili (Seri $n$=4) olarak çalıştırmaktayız.

PACO-3Opt algoritması ve seri hali aynı problemler üzerinde çalıştırıldığından dolayı ve elde edilen sonuçların dağılımı hakkında yeterli bilgiye sahip olunmadığı için

parametrik olmayan istatistik testleri kullanılmıştır. Küçük ölçekli problemlerin sonuçlarının kıyaslanmasında karşılaştırılacak iki tane anakütle olduğu için Wilcoxon sıra toplam testi kullanılmıştır. Büyük ölçekli problemlerin sonuçlarının kıyaslanmasında ise karşılaştırılacak dört tane anakütle (PACO-3Opt $n$=2, PACO-3Opt $n$=4, Seri $n$=2 ve Seri $n$=4) olduğu için Kruskal-Wallis testi kullanılmıştır.

Küçük ölçekli problemlerde, PACO-3Opt algoritması seri versiyonu ile detaylı olarak karşılaştırılmıştır ve elde edilen en iyi, en kötü ve ortalama tur uzunluğu sonuçları Çizelge 5.37'de verilmiştir. Çizelge 5.37'e göre, PACO-3Opt algoritması ve seri hali bütün problemlerde yakın sonuçlar elde etmişlerdir. Ayrıca Çizelge 5.37'deki son sütun, bu iki algoritmanın farklı olup olmadığını istatistiksel olarak belirleyen Wilcoxon sıra toplam testinin $p$ değerlerini göstermektedir.

PACO-3Opt algoritmasının ve seri versiyonunun sonuçlarına göre bu iki algoritma arasında 95% ($\alpha$=0.05) önem seviyesinde istatistiksel olarak bir fark olup olmadığını belirlemek için $H_0$ ve $H_1$ hipotezleri şu şekilde kurulabilir:

$H_0$: PACO-3Opt algoritması ve seri versiyonu arasında farklılık yoktur.

$H_1$: PACO-3Opt algoritması ve seri versiyonu birbirinden farklıdır. (çift yönlü test)

Hipotezin kabul edilip edilmeyeceğine Wilcoxon sıra toplam testinin $p$ değerine göre karar verilmektedir. Eğer $p$ değeri 0.05'den küçük ise o zaman $H_0$ hipotezi 95% önem seviyesinde reddedilir ve $H_1$ hipotezi kabul edilir. Yani, bu iki algoritma 95% önem seviyesinde birbirinden farklıdır. Tersi durumunda, eğer $p$ değeri 0.05'den büyük ise o zaman $H_0$ hipotezi 95% önem seviyesinde kabul edilir ve $H_1$ hipotezi reddedilir. Bu durumda, 95% önem seviyesinde iki algoritma arasında farklılık yoktur. Wilcoxon sıra toplam testi, PACO-3Opt algoritmasının ve seri halinin sonuçları üzerinde çalıştırılmıştır. Elde edilen $p$ değerleri Çizelge 5.37'de gösterilmektedir. $p$ değerlerine göre $H_0$ hipotezi tüm problemler için kabul edilmektedir. Bu verilerden yola çıkarak, 95% önem seviyesinde istatistiksel olarak PACO-3Opt algoritması ve seri versiyonu arasında farklılık olmadığı sonucuna varılmaktadır.

Algoritmaların hesaplama süreleri Çizelge 5.38'da saniye cinsinden verilmiştir. Hesaplama süresi açısından karşılaştıracak olursak, PACO-3Opt algoritması mükemmel bir performans sergilemiştir. Çizelge 5.38'daki sonuçlardan görüldüğü gibi, PACO-3Opt algoritması bütün test örneklerini seri versiyonundan daha hızlı çözmektedir.

**Çizelge 5.37.** Küçük ölçekli problemlerde tur uzunluğu açısından algoritmaların karşılaştırılması ve Wilcoxon sıra toplam testine göre *p* değeri

| Örnek | En iyi tur uzunluğu | | En kötü tur uzunluğu | | Ortalama tur uzunluğu | | *p* değeri |
|---|---|---|---|---|---|---|---|
| | PACO-3Opt | Seri | PACO-3Opt | Seri | PACO-3Opt | Seri | |
| eil51 | 426 | 426 | 427 | 427 | 426.35 | 426.45 | 0.602 |
| berlin52 | 7542 | 7542 | 7542 | 7542 | 7542.00 | 7542.00 | 1.000 |
| rat99 | 1213 | 1213 | 1225 | 1224 | 1217.10 | 1217.20 | 0.799 |
| eil76 | 538 | 538 | 542 | 542 | 539.85 | 539.45 | 0.289 |
| st70 | 676 | 676 | 679 | 679 | 677.85 | 677.60 | 0.369 |
| kroA100 | 21282 | 21309 | 21382 | 21391 | 21326.80 | 21333.65 | 0.369 |
| lin105 | 14379 | 14379 | 14422 | 14433 | 14393.00 | 14390.95 | 0.925 |
| kroA200 | 29533 | 29526 | 29721 | 29750 | 29644.50 | 29646.65 | 0.883 |
| ch150 | 6570 | 6570 | 6627 | 6618 | 6601.40 | 6592.65 | 0.102 |
| eil101 | 629 | 629 | 639 | 639 | 630.55 | 631.10 | 0.779 |

**Çizelge 5.38.** Küçük ölçekli problemlerde hesaplama süresi açısından algoritmaların karşılaştırılması

| Örnek | Süre (s) | |
|---|---|---|
| | PACO-3Opt | Seri |
| eil51 | 2.39 | 7.18 |
| berlin52 | 2.10 | 7.09 |
| rat99 | 19.79 | 61.61 |
| eil76 | 8.18 | 23.60 |
| st70 | 6.97 | 21.36 |
| kroA100 | 21.10 | 68.96 |
| lin105 | 14.57 | 66.51 |
| kroA200 | 213.12 | 743.77 |
| ch150 | 79.35 | 253.80 |
| eil101 | 20.79 | 70.84 |

Büyük ölçekli problemlerde, PACO-3Opt algoritması ve seri versiyonu 2 kolonili ($n$=2) olarak ve 4 kolonili ($n$=4) olarak çalıştırılmıştır ve elde edilen ortalama, en iyi ve en kötü tur uzunlukları sırasıyla Çizelge 5.39, Çizelge 5.40 ve Çizelge 5.41'da verilmiştir. Çizelge 5.39, Çizelge 5.40 ve Çizelge 5.41'a göre, bu dört algoritmanın sonuçları bütün problemlerde birbirine yakın çıkmaktadır. Ayrıca Çizelge 5.39'deki son sütun, bu dört algoritmanın farklı olup olmadığını istatistiksel olarak belirleyen Kruskal-Wallis testinin *p* değerlerini göstermektedir.

Dört tane bağımsız örnek olduğu için bu örneklerin farklı anakütlelerden gelip gelmediğine karar vermede Kruskal-Wallis istatistiksel yöntemi kullanılmıştır. Elde edilen sonuçlarına göre bu dört algoritma arasında 95% ($\alpha$=0.05) önem seviyesinde istatistiksel olarak bir fark olup olmadığını belirlemek için $H_0$ ve $H_1$ hipotezleri şu şekildedir:

$H_0$: PACO-3Opt $n$=2, PACO-3Opt $n$=4, Seri $n$=2 ve Seri $n$=4 algoritmaları aynıdır.

$H_1$: PACO-3Opt $n$=2, PACO-3Opt $n$=4, Seri $n$=2 ve Seri $n$=4 algoritmaları farklıdır.

Hipotezin kabul edilip edilmeyeceğine Kruskal-Wallis testinin *p* değerine göre karar verilmektedir. Eğer *p* değeri 0.05'den küçük ise o zaman $H_0$ hipotezi 95% önem seviyesinde reddedilir ve $H_1$ hipotezi kabul edilir. Yani, bu algoritmalar 95% önem seviyesinde birbirinden farklıdır. Tersi durumunda, eğer *p* değeri 0.05'den büyük ise o zaman $H_0$ hipotezi 95% önem seviyesinde kabul edilir ve $H_1$ hipotezi reddedilir. Bu durumda, 95% önem seviyesinde bu algoritmalar arasında farklılık yoktur. Kruskal-Wallis testi algoritmaların sonuçları üzerinde çalıştırılmıştır. Elde edilen *p* değerleri Çizelge 5.39'de gösterilmektedir. *p* değerlerine göre $H_0$ hipotezi pr439, pcb442 ve d657 hariç diğer tüm problemler için kabul edilmektedir. Bu verilerden yola çıkarak, bu algoritmaların aynı olduklarına 95% önem seviyesinde karar verilir.

**Çizelge 5.39.** Büyük ölçekli problemlerde ortalama tur uzunluğu açısından algoritmaların karşılaştırılması ve Kruskal-Wallis testine göre *p* değeri

| Örnek | BES | Ortalama tur uzunluğu | | | | *p* değeri |
|---|---|---|---|---|---|---|
| | | PACO-3Opt n=2 | PACO-3Opt n=4 | Seri n=2 | Seri n=4 | |
| rd400 | 15281 | 15601.0 | 15613.9 | 15628.4 | 15609.5 | 0.252 |
| fl417 | 11861 | 11991.9 | 11987.4 | 11994.0 | 11989.0 | 0.181 |
| pr439 | 107217 | 108736.6 | 108702.0 | 108794.5 | 108723.1 | 0.010 |
| pcb442 | 50778 | 52267.5 | 52202.4 | 52315.8 | 52207.7 | 0.019 |
| d493 | 35002 | 35811.0 | 35841.0 | 35842.9 | 35849.6 | 0.821 |
| u574 | 36905 | 38020.3 | 38030.7 | 38036.4 | 38029.9 | 0.901 |
| rat575 | 6773 | 7024.5 | 7012.4 | 7024.4 | 7019.7 | 0.109 |
| p654 | 34643 | 35065.8 | 35075.0 | 35095.1 | 35086.0 | 0.051 |
| d657 | 48912 | 50352.9 | 50277.5 | 50381.8 | 50302.3 | 0.003 |
| u724 | 41910 | 43100.1 | 43122.5 | 43158.8 | 43146.1 | 0.709 |
| rat783 | 8806 | 9130.7 | 9127.3 | 9131.1 | 9128.0 | 0.564 |

BES: bilinen en iyi sonuç

Algoritmaların hesaplama süreleri saniye cinsinden Çizelge 5.42'da verilmiştir. Hesaplama süresi açısından karşılaştıracak olursak, algoritmanın paralel hali seri haline göre mükemmel bir performans sergilemiştir. Çizelge 5.42'daki sonuçlardan görüldüğü gibi, PACO-3Opt algoritması bütün test örneklerini seri versiyonundan oldukça daha hızlı çözmektedir.

**Çizelge 5.40.** Büyük ölçekli problemlerde en iyi tur uzunluğu açısından algoritmaların karşılaştırılması

| Örnek | BES | En iyi tur uzunluğu | | | |
|---|---|---|---|---|---|
| | | PACO-3Opt n=2 | PACO-3Opt n=4 | Seri n=2 | Seri n=4 |
| rd400 | 15281 | 15527 | 15578 | 15558 | 15546 |
| fl417 | 11861 | 11958 | 11972 | 11965 | 11982 |
| pr439 | 107217 | 108516 | 108482 | 107743 | 108488 |
| pcb442 | 50778 | 52057 | 51962 | 52087 | 52027 |
| d493 | 35002 | 35742 | 35735 | 35777 | 35737 |
| u574 | 36905 | 37855 | 37981 | 37900 | 37919 |
| rat575 | 6773 | 7009 | 7003 | 7000 | 6997 |
| p654 | 34643 | 35027 | 35045 | 35014 | 35041 |
| d657 | 48912 | 50230 | 50206 | 50212 | 50221 |
| u724 | 41910 | 42893 | 42764 | 42921 | 43059 |
| rat783 | 8806 | 9112 | 9111 | 9092 | 9112 |

BES: bilinen en iyi sonuç

**Çizelge 5.41.** Büyük ölçekli problemlerde en kötü tur uzunluğu açısından algoritmaların karşılaştırılması

| Örnek | BES | En kötü tur uzunluğu | | | |
|---|---|---|---|---|---|
| | | PACO-3Opt n=2 | PACO-3Opt n=4 | Seri n=2 | Seri n=4 |
| rd400 | 15281 | 15663 | 15667 | 15688 | 15676 |
| fl417 | 11861 | 12005 | 12000 | 12014 | 12003 |
| pr439 | 107217 | 108889 | 108973 | 109138 | 108965 |
| pcb442 | 50778 | 52393 | 52283 | 52559 | 52357 |
| d493 | 35002 | 35927 | 35982 | 35945 | 35920 |
| u574 | 36905 | 38157 | 38165 | 38153 | 38178 |
| rat575 | 6773 | 7041 | 7037 | 7063 | 7044 |
| p654 | 34643 | 35123 | 35116 | 35158 | 35120 |
| d657 | 48912 | 50480 | 50386 | 50489 | 50418 |
| u724 | 41910 | 43240 | 43272 | 43328 | 43216 |
| rat783 | 8806 | 9145 | 9152 | 9155 | 9153 |

BES: bilinen en iyi sonuç

**Çizelge 5.42.** Büyük-ölçekli problemler üzerinde algoritmaların hesaplama süreleri (s)

| Örnek | PACO-3Opt n=2 | PACO-3Opt n=4 | Seri n=2 | Seri n=4 |
|---|---|---|---|---|
| rd400 | 2841 | 1496 | 9195 | 9007 |
| fl417 | 4027 | 2046 | 11693 | 12038 |
| pr439 | 4231 | 2132 | 12841 | 13151 |
| pcb442 | 3879 | 2088 | 11706 | 12197 |
| d493 | 6248 | 3175 | 18623 | 19122 |
| u574 | 10654 | 5460 | 31848 | 33390 |
| rat575 | 9751 | 5004 | 27182 | 25135 |
| p654 | 17681 | 9105 | 48137 | 51851 |
| d657 | 17088 | 8715 | 51230 | 47351 |
| u724 | 23282 | 11458 | 70152 | 61055 |
| rat783 | 28657 | 14277 | 86408 | 84964 |

# 6. SONUÇLAR VE ÖNERİLER

## 6.1 Sonuçlar

Bilim ve teknolojideki hızlı ilerlemeler her alanda olduğu gibi hesaplama alanındaki gelişmeleri de tetiklemektedir. Bu gelişmelerin sonucu olarak, bilgisayarlar tarafından çözülmesi gereken problemler daha büyük ve daha karmaşık olmaktadır ve büyük ölçekli verilerin artmasına neden olmaktadır. Böylece, bu problemleri çözmek ve büyük ölçekli verileri işlemek için paralel hesaplamaya ve paralel algoritmalara ihtiyaç duyulmaktadır. Paralel hesaplamada, sonuçları daha kısa sürede elde etmek için alt görevlere ayrılmış bir görev çoklu işlemcilerde eşzamanlı olarak çalıştırılır. Paralel hesaplama kullanılarak, algoritmanın performansı artmaktadır ve büyük ölçekli problemler daha kısa zamanda çözülmektedir.

Optimizasyon matematik, finans, bilgisayar bilimi ve mühendisliği gibi çeşitli alanlardaki problemlerin çözümü için önemli bir tekniktir. Optimizasyon problemleri genel olarak *sürekli* ve *ayrık* olarak iki gruba ayrılmaktadır. Ayrık optimizasyon, değişkenlerin sadece ayrık değer (genellikle tamsayı) alabildiği durumlarla ilgilenmektedir. Bu problemler genellikle çok zordur ve bütün olası çözümlerin teker teker denenmesi ile en iyi çözümün bulunduğu garanti edilmektedir. Ayrık optimizasyonun aksine sürekli optimizasyonda amaç fonksiyonunda kullanılan değişkenlerin, arasında boşluk bulunmayan gerçek değerler kümesinden seçilen sürekli değişkenler olmaları gerekmektedir. Optimizasyon problemlerini çözmek için birçok metasezgisel algoritma geliştirilmiştir ve yeni algoritmalar da önerilmektedir. Metasezgisel optimizasyon algoritmaları genellikle sıralı bir yapıya sahiptir. Metasezgisel optimizasyon algoritmalarının kullanımı problemin çözüm süresini düşürmesine rağmen bazı gerçek dünya problemlerinin çözümleri hala çok uzun zaman almaktadır. Bu yüzden, hem çözüm süresini azaltmak hem de çözümlerin kalitesini artırmak için paralel hesaplamanın metasezgisel algoritmalar ile beraber kullanılması araştırmacıların ilgi duyduğu bir konu olmaktadır.

Bu tez çalışmasında, sürekli optimizasyon probleminin çözümü için PSO temelli ve ayrık optimizasyon probleminin çözümü için ACO temelli iki yeni paralel metasezgisel algoritma geliştirilmiştir.

Önerilen paralel algoritmaları geliştirmek için bir arakatman olan Jade yazılım çatısı kullanılmıştır. Arakatman; farklı yapıdaki bilgisayarları, ağları ve işletim

sistemlerini destekleyen ve ayrıca paralel uygulamalar geliştirmeyi mümkün kılan işletim sistemleri ve uygulamalar arasındaki bir yazılım katmanıdır. Böylece, dağıtık sistem üzerinde paralel programlamayı mümkün kılan Jade programlama çatısı tezde geliştirilen algoritmaların alt yapısını oluşturmaktadır.

Bu tez çalışmasında Açgözlü Bilgi Takası (ABT) diye adlandırılan yeni bir işbirliği stratejisi geliştirilmiştir ve literatüre kazandırılmıştır. Geliştirilen iki paralel metasezgisel algoritma ABT işbirliği stratejisini kullanmaktadır. ABT'nin başarısını etkileyen dört faktör bulunmaktadır: göç periyodu, topoloji, takas edilecek bilgi ve yer değiştirme poliçesi. Göç periyodunu belirlemenin iki çeşit stratejisi vardır: *Kör* ve *adaptif*. Bu çalışmada, kör stratejisi tercih edilmiştir. Bu stratejide göç süreci, periyodik olarak belirli bir iterasyon sayısı sonrasında oluşmaktadır. İkinci faktör bilgileri paylaşmak için komşu sürülerin belirlendiği topolojidir. ABT topolojisinde haberleşmeler sadece sunucu-sürü ile istemci sürüler arasındadır ve istemci sürüler arasında doğrudan bir haberleşme bulunmamaktadır. Bu topolojide düşük haberleşme frekansı olduğu için yüksek hesaplama verimliliği elde edilmektedir. Üçüncü faktör sürüler arasında takas edilecek bilgidir. ABT stratejisinde, her bir sürü kendi lokal en iyi çözümünü paylaşmaktadır. Dördüncü faktör alınan bilgilerin nasıl kullanılacağıyla ilgilenen yer değiştirme poliçesidir. Bu aşamada her bir istemci sürüden rasgele seçilen birer parçacık, sunucu tarafından gönderilmiş olan global en iyi parçacık ile değiştirilmektedir.

Tez çalışması iki kısımdan oluşmaktadır. Tezin ilk kısmında, sürekli optimizasyon problemlerin bir alt dalı olan tek amaçlı optimizasyon test problemlerinin çözümü için PCLPSO algoritması geliştirilmiştir. PCLPSO algoritması, CLPSO algoritmasının paralelleştirilmiş şeklidir. PCLPSO algoritmasının sağladığı avantajlar şu şekilde sıralanabilir:

- PCLPSO algoritması çoklu-sürü ve işbirliği özellikleri sayesinde çözüm kalitesini ve global arama kabiliyetini artırmaktadır.
- Çoklu-sürü tekniğinde, popülasyon alt popülasyonlara bölünmektedir ve her bir sürü birbirinden bağımsız olarak tüm arama uzayında çalışmaktadır.
- Sürüler en güncel çözümleri takas etmek için ve aramayı daha kaliteli çözümlere doğru yönlendirmek için periyodik olarak işbirliği yapmaktadır.
- PCLPSO algoritması dağıtık sistem üzerinde paralel olarak çalışması sayesinde aramayı önemli derecede hızlandırmaktadır.

- Büyük ölçekli problemleri makul süreler içerisinde çözmektedir.

Önerilen yöntem, 14 tane kıyaslama fonksiyonu üzerinde test edilmiştir. Bu fonksiyonlar global optimizasyon üzerine çalışanlar tarafından iyi bilinmektedirler ve genellikle optimizasyon algoritmalarının testi için kullanılmaktadırlar. Önerilen algoritmanın sonuçları literatürdeki PSO tabanlı 9 farklı algoritmanın sonuçları ile kıyaslanmıştır.

Deneyler fonksiyonların boyutlarına göre üç gruba bölünmüştür. Bütün testleri diğer algoritmalar ile eşit şartlarda yapmak için popülasyon boyutu ve maksimum uygunluk fonksiyonu değerlendirmesi (FE) bütün algoritmalarda aynı sayıda tutulmaktadır. İlk deney grubunda, fonksiyonlar 10 boyutludur ve PCLPSO algoritmasında bir sunucu ve bir istemci olmak üzere iki sürü kullanılmaktadır. İkinci deney grubunda, fonksiyonlar 30 boyutludur ve PCLPSO algoritmasında bir sunucu ve üç istemci olmak üzere dört sürü kullanılmaktadır. Üçüncü deney grubunda, fonksiyonlar 100 boyutludur ve PCLPSO algoritmasında bir sunucu ve üç istemci olmak üzere dört sürü kullanılmaktadır. Her bir fonksiyon için 30 bağımsız çalıştırma uygulanmaktadır. Deneyler dağıtık sistem kümesi üzerinde gerçekleştirilmiştir. Küme 10 boyutlu problemler için iki bilgisayara, 30 ve 100 boyutlu problemler için ise dört bilgisayara sahiptir. Göç periyodu $P$ bütün deney gruplardaki test fonksiyonların çözümünde 20 olarak alınmaktadır.

Sonuçlar ortalama ve standart sapma değerleri açısından karşılaştırılmıştır ve PCLPSO algoritmasının diğer algoritmalardan daha başarılı olduğunu göstermiştir. Ayrıca, CLPSO ve PCLPSO algoritmalarının karşılaştırılmasında Wilcoxon sıra toplam testi kullanılmıştır. Wilcoxon sıra toplam testine göre, 95% önem seviyesinde istatistiksel olarak CLPSO ve PCLPSO algoritmalarının birbirinden farklı olduğu sonucuna varılmıştır. Bu iki algoritmayı hesaplama süresi açısından karşılaştırdığımızda, PCLPSO büyük ölçekli problemlerde çok iyi performans göstermiştir. Boyut sayısı arttıkça PCLPSO problemleri CLPSO'dan oldukça daha hızlı çözmektedir.

PCLPSO algoritması seri versiyonu ile de kıyaslanmıştır. Ortalama ve standart sapma değerleri açısından karşılaştırdığımız zaman, PCLPSO algoritması ve seri hali bütün fonksiyonlarda yakın sonuçlar elde etmişlerdir. Wilcoxon sıra toplam testine göre, 95% önem seviyesinde istatistiksel olarak PCLPSO algoritması ve seri hali arasında farklılık olmadığı sonucuna varılmıştır. Hesaplama süresi açısından

karşılaştırdığımız zaman, problem boyutu arttıkça PCLPSO problemleri seri versiyonundan oldukça daha hızlı çözmüştür.

Ayrıca, ABT stratejisinin topolojisi halka, 2B ağ ve tam çizge topolojileri ile karşılaştırılmıştır. PCLPSO algoritmasının başarısını etkileyen önemli faktörlerden bir tanesi şüphesiz ki kullanılan topolojidir. Topolojilerin karşılaştırılması, elde edilen sonuçların ortalama değerleri ve ortalama hesaplama süresi üzerinden yapılmıştır. Sonuçlara göre, ABT'nin topolojisi 14 fonksiyonun 10'unda en iyi ortalama sonuca sahiptir. Geriye kalan 4 fonksiyonda ise en iyi ortalama sonuca tam çizge topolojisi sahip olmuştur. ABT ve tam çizge topolojilerinden sonra en iyi sonuçlara 2B ağ topolojisi sahip olmuştur ve en kötü sonuçlar halka topolojisinde elde edilmiştir. ABT topolojisinde haberleşmeler sadece sunucu-sürü ile istemci sürüler arasındadır ve istemci sürüler arasında doğrudan bir haberleşme bulunmamaktadır. Bundan dolayı bu topolojide düşük haberleşme frekansı olduğu için yüksek hesaplama verimliliği elde edilmektedir. Hesaplama süresi açısından incelendiğinde, 14 fonksiyon için ABT'nin topolojisi diğer topolojilerden daha iyi sonuçlar ürettiği görülmektedir. Genel olarak deney sonuçlarına bakıldığında hem hesaplama süresi hem de çözümlerin kalitesi açısından, ABT'nin topolojisinin diğer topolojilerden daha başarılı sonuçlar ürettiği görülmektedir.

Tezin ikinci kısmında, ayrık optimizasyon problemi olan GSP çözümü için PACO-3Opt (paralel karınca koloni optimizasyonu ile birleştirilmiş 3Opt algoritması) algoritması geliştirilmiştir. PACO-3Opt paralel metasezgisel algoritması da ABT işbirliği stratejisini kullanmaktadır.

GSP tam sonucu veren algoritmalar veya sezgisel yöntemler kullanılarak çözülebilmektedir. Tam sonucu veren algoritmalar büyük ölçekli GSP için uygun değildir. Birkaç sezgisel algoritma büyük ölçekli GSP'nin çözümü için geliştirilmiştir. Sezgisel yaklaşımlar GSP'nin çözümünde iyi sonuçlar elde etmelerine rağmen, lokal optimumlardan başarılı şekilde kaçamamaktadırlar. GSP'nin çözümünde etkili algoritmalardan birisi de şüphesiz ki ACO algoritmasıdır. Fakat tek koloniye sahip ACO algoritmasının en büyük eksikliği aramanın erken durgunluğudur. Bunun yanında, GSP'nin çözümünde sezgisel yaklaşımların çalışma süreleri ne yazık ki çok uzun zaman almaktadır. Bu eksikliklerin üstesinden gelmek ve çözüm kalitesini artırmak için 3-Opt lokal arama algoritmasını kullanan karınca koloni optimizasyonuna dayalı çoklu-koloniye sahip ve paralel çalışan paralel işbirlikçi hibrit bir algoritma (PACO-3Opt) geliştirilmiştir ve literatüre kazandırılmıştır. PACO-3Opt algoritması onun çoklu-koloni

ve işbirliği özelliği sayesinde çözümlerin kalitesini ve kararlılığı artırmaktadır. Dağıtık hesaplama ortamında paralel olarak çalışma yeteneğiyle hesaplama süresini düşürmektedir. Ayrıca, 3-Opt lokal arama algoritması ile her bir koloni tarafından bulunan turlar iyileştirilmektedir.

PACO-3Opt algoritması, hem küçük ölçekli problemler üzerinde hem de büyük ölçekli problemler üzerinde test edilmiştir. Küçük ölçekli problem olarak şehir sayısı 50 ile 200 arasında olan 10 adet simetrik GSP örneği kullanılmıştır ve sonuçlar 14 farklı algoritmanın sonuçları ile kıyaslanmıştır. Büyük ölçekli problem olarak ise şehir sayısı 400 ile 800 arasında olan 11 adet simetrik GSP örneği kullanılmıştır ve sonuçlar 8 farklı algoritmanın sonuçları ile kıyaslanmıştır. Sonuçlar PACO-3Opt algoritmasının küçük-ölçekli ve büyük-ölçekli GSP örneklerinin çözümünde iyi bir performans elde ettiğini göstermiştir. PACO-3Opt algoritmasının karşılaştırılan algoritmalardan daha tutarlı ve daha iyi sonuçlar elde ettiği gözlemlenmiştir. PACO-3Opt algoritması diğer optimizasyon algoritmaları ile oldukça rekabetçidir ve GSP örneklerini çözen herhangi bir meta sezgisel algoritmaya alternatif olabilir. Hesaplama süresi açısından karşılaştıracak olursak, PACO-3Opt algoritması mükemmel bir performans sergilemiştir. PACO-3Opt algoritması bütün test örneklerini PSO-ACO-3Opt algoritmasından oldukça daha hızlı çözmektedir.

Algoritmaları kıyaslamak için kullanılan bir diğer yöntem ise kutu grafiğidir. Bu yöntem ile algoritmaların başarıları ile ilgili istatistiksel bilgiler görselleştirilerek, bu bilgilerin yorumlanması kolaylaştırılır. Kutu grafiği veri kalitesi hakkında birçok bilgiyi bir bakışta iletmek için mükemmeldir. Karşılaştırılacak verilerin kalitesini ve dağılımını belirlemek için kullanılır. Ayrıca kutu grafiği algoritmanın istikrarı ve gürbüzlüğü hakkında bilgi vermektedir. PACO-3Opt ve PSO-ACO-3Opt algoritmalarının küçük ölçekli problemler üzerinde elde ettiği sonuçların kutu grafiği incelendiğinde, PACO-3Opt algoritması 3 örnek (eil76, lin105 ve ch150) dışındaki tüm örneklerde PSO-ACO-3Opt algoritmasından daha iyi sonuçlar üretmiştir. Ayrıca, birinci çeyrek ve üçüncü çeyrek arasında kalan alan açısından grafik incelendiğinde, 3 örnek (rat99, eil76 ve lin105) hariç diğer tüm örneklerde PACO-3Opt'un daha dar bir alan ürettiği gözükmektedir. PACO-3Opt ve PSO-ACO-3Opt algoritmalarının büyük ölçekli problemler üzerinde elde ettiği sonuçların kutu grafiği incelendiğinde, PACO-3Opt algoritması 11 örneğin 10'unda PSO-ACO-3Opt algoritmasından daha iyi sonuçlar elde etmiştir. PSO-ACO-3Opt algoritması sadece fl417 örneğinde başarılıdır. Ayrıca, en küçük değer ile en büyük değer arasında kalan alan açısından grafik incelendiğinde, 2

örnek (u724 ve rat783) dışındaki diğer tüm örneklerde PACO-3Opt'un daha dar bir alan ürettiği gözükmektedir. Bundan dolayı, PACO-3Opt algoritmasının daha istikrarlı olduğu söylenebilir.

Ayrıca, PACO-3Opt algoritması seri versiyonu ile kıyaslanmıştır. PACO-3Opt algoritması bütün test örneklerini seri versiyonundan oldukça daha hızlı çözmektedir. PACO-3Opt algoritması ile seri versiyonunun birbirinden farklı olup olmadığını araştırmak için istatistiksel yöntemlerden faydalanılmıştır. Küçük ölçekli problemlerin sonuçlarının kıyaslanmasında karşılaştırılacak iki tane anakütle olduğu için Wilcoxon sıra toplam testi, büyük ölçekli problemlerin sonuçlarının kıyaslanmasında ise karşılaştırılacak dört tane anakütle olduğu için Kruskal-Wallis testi kullanılmıştır. Küçük ölçekli problemlerde $H_0$ hipotezi tüm problemler için kabul edilmektedir. Büyük ölçekli problemlerde $H_0$ hipotezi pr439, pcb442 ve d657 hariç diğer tüm problemler için kabul edilmektedir. Bu verilerden yola çıkarak, istatistiksel olarak 95% önem seviyesinde bu algoritmaların aynı oldukları sonucuna varılmaktadır.

Genel olarak PCLPSO algoritmasının ve PACO-3Opt algoritmasının deneysel sonuçlarını incelediğimizde, bu algoritmalar birçok problemde karşılaştırılan algoritmalardan daha iyi çözümler elde etmişlerdir. Ayrıca sadece az sayıdaki problemde kötü çözümler elde etmişlerdir ki bu durum *no free lunch teoremine* göre tamamen normaldir ve kabul edilebilir. *No free lunch* teoremine göre bütün problemleri en iyi şekilde çözecek tek bir optimizasyon algoritmasının olması mümkün değildir. Bir algoritma birçok problemde iyi sonuçlar vermesine rağmen bazı problemlerde kötü sonuçlar elde etmesi normal olarak karşılanmaktadır.

Sonuç olarak, paralel hesaplama kullanılarak optimizasyon algoritmalarının performansı artırılmıştır. Bu bağlamda PCLPSO ve PACO-3Opt paralel metasezgisel algoritmaları ve ABT işbirliği stratejisi geliştirilmiştir.

## 6.2 Öneriler

Geliştirilen paralel metasezgisel algoritmaların performansını etkileyen bazı faktörler vardır. İleri çalışma olarak, sürü sayısının algoritmanın performansı üzerindeki etkileri araştırılmalıdır. Ayrıca göç aralığının ve sürüler arasında takas edilen bilgilerin algoritmanın performansı üzerindeki etkileri de araştırılmalıdır.

Ayrıca, göç sürecinin ne zaman gerçekleştirileceğini adaptif olarak belirleyen yöntem geliştirilebilir.

Geliştirilen PCLPSO ve PACO-3Opt algoritmaları çoklu optimizsasyon problemleri, asimetrik GSP örnekleri (Cirasella ve ark., 2001), genelleştirilmiş GSP örnekleri (Snyder ve Daskin, 2006) gibi farklı problem türlerine de uygulanabilir.

Paralel metasezgisel algoritma geliştirmedeki mevcut eğilim GPU'ların kullanımına doğru gitmektedir (Tsutsui ve Collet, 2013). Bundan dolayı, PCLPSO ve PACO-3Opt algoritmalarının GPU üzerinde çalışan versiyonları geliştirilebilir.

**KAYNAKLAR**

Acılar, A. M., 2013, The fuzzy system designing using artificial immune system algorithms, Ph.D. Thesis, *Selçuk University, the graduate school of natural and applied science*, Konya.

Alba, E., 2005, Measuring the Performance of Parallel Metaheuristics In: Parallel metaheuristics: a new class of algorithms, *John Wiley & Sons*, Hoboken, New Jersey.

Alpar, R., 2012, Uygulamalı istatistik ve geçerlik-güvenirlik: spor, sağlık ve eğitim bilimlerinden örneklerle, *Detay Yayıncılık*.

Atiqullah, M. M. ve Rao, S., 2000, Simulated annealing and parallel processing: an implementation for constrained global design optimization, *Engineering Optimization*, 32 (5), 659-685.

Avşar, B. ve Aliabadi, D. E., 2015, Parallelized neural network system for solving Euclidean traveling salesman problem, *Applied Soft Computing*, 34, 862-873.

Balakrishnan, N., Gopinatha, J., Goswami, D. ve Shanker, L., 2014, Parallel computing strategies in the analysis of the inhibiting effect of price limits on futures prices, *Concurrency and Computation: Practice and Experience*, 26 (9), 1666-1678.

Bansal, J., Singh, P., Saraswat, M., Verma, A., Jadon, S. S. ve Abraham, A., 2011, Inertia weight strategies in particle swarm optimization, *Third World Congress on Nature and Biologically Inspired Computing (NaBIC 2011)*, 633-640.

Bellifemine, F. L., Caire, G. ve Greenwood, D., 2007, Developing multi-agent systems with JADE, *John Wiley & Sons*, England.

Bui, P., Boettcher, T., Jaeger, N. ve Westphal, J., 2013, Using clusters in undergraduate research: Distributed animation rendering, photo processing, and image transcoding, *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, 1-8.

Calazan, R. M., Nedjah, N. ve Mourelle, L. M., 2014, A hardware accelerator for particle swarm optimization, *Applied Soft Computing*, 14, 347-356.

Chaves-González, J. M., Vega-Rodríguez, M. A., Gómez-Pulido, J. A. ve Sánchez-Pérez, J. M., 2011, Optimizing a realistic large-scale frequency assignment problem using a new parallel evolutionary approach, *Engineering Optimization*, 43 (8), 813-842.

Chen, H., Zhu, Y. ve Hu, K., 2011, Adaptive bacterial foraging optimization, *Abstract and Applied Analysis*, 2011, 1-27.

Chen, M.-R., Li, X., Zhang, X. ve Lu, Y.-Z., 2010, A novel particle swarm optimizer hybridized with extremal optimization, *Applied Soft Computing*, 10 (2), 367-373.

Chen, S.-M. ve Chien, C.-Y., 2011, Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques, *Expert Systems with Applications*, 38 (12), 14439-14450.

Chen, X., Huang, X., Jiao, C., Flanner, M. G., Raeker, T. ve Palen, B., 2017, Running climate model on a commercial cloud computing environment: A case study using Community Earth System Model (CESM) on Amazon AWS, *Computers & Geosciences*, 98, 21-25.

Chica, M., Cordón, Ó., Damas, S. ve Bautista, J., 2015, Interactive preferences in multiobjective ant colony optimisation for assembly line balancing, *Soft Computing*, 19 (10), 2891-2903.

Cirasella, J., Johnson, D. S., McGeoch, L. A. ve Zhang, W., 2001, The asymmetric traveling salesman problem: Algorithms, instance generators, and tests, In: Algorithm Engineering and Experimentation, Eds: Buchsbaum, A. L. ve Snoeyink, J., *Springer*, Berlin, Heidelberg.

Clerc, M. ve Kennedy, J., 2002, The particle swarm-explosion, stability, and convergence in a multidimensional complex space, *IEEE Transactions on Evolutionary Computation*, 6 (1), 58-73.

Correa, E. S., Freitas, A. A. ve Johnson, C. G., 2006, A new discrete particle swarm algorithm applied to attribute selection in a bioinformatics data set, *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 35-42.

Delévacq, A., Delisle, P., Gravel, M. ve Krajecki, M., 2013, Parallel ant colony optimization on graphics processing units, *Journal of Parallel and Distributed Computing*, 73 (1), 52-61.

Dolan, E. D., Moré, J. J. ve Munson, T. S., 2004, Benchmarking optimization software with COPS 3.0, *Argonne National Laboratory Technical Report ANL/MCS-TM-273*.

Donald, D., 2011, Traveling salesman problem, theory and applications, *InTech*, Rijeka.

Dong, G., Guo, W. W. ve Tickle, K., 2012, Solving the traveling salesman problem using cooperative genetic ant systems, *Expert Systems with Applications*, 39 (5), 5006-5011.

Dorigo, M., Maniezzo, V. ve Colorni, A., 1996, Ant system: optimization by a colony of cooperating agents, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26 (1), 29-41.

Dorigo, M. ve Gambardella, L. M., 1997, Ant colony system: a cooperative learning approach to the traveling salesman problem, *IEEE Transactions on Evolutionary Computation*, 1 (1), 53-66.

Dorigo, M. ve Stützle, T., 2004, Ant colony optimization, *MIT Press*, Cambridge, Mass.

Duan, H., Ma, G. ve Liu, S., 2007, Experimental study of the adjustable parameters in basic ant colony optimization algorithm, *IEEE Congress on Evolutionary Computation CEC 2007*, 149-156.

Esquivel, S. C. ve Coello Coello, C. A., 2003, On the use of particle swarm optimization with multimodal functions, *The 2003 Congress on Evolutionary Computation CEC'03*, 1130-1136.

Fan, S.-K. S. ve Chang, J.-M., 2009, A parallel particle swarm optimization algorithm for multi-objective optimization problems, *Engineering Optimization*, 41 (7), 673-697.

Fan, S.-K. S. ve Chang, J.-M., 2010, Dynamic multi-swarm particle swarm optimizer using parallel PC cluster systems for global optimization of large-scale multimodal functions, *Engineering Optimization*, 42 (5), 431-451.

Feng, X., Lau, F. C. ve Yu, H., 2013, A novel bio-inspired approach based on the behavior of mosquitoes, *Information Sciences*, 233, 87-108.

Ferreira, C., 2006, Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence (Studies in Computational Intelligence), *Springer-Verlag New York, Inc.*, Heidelberg.

Flynn, M. J., 1972, Some computer organizations and their effectiveness, *IEEE transactions on computers*, 100 (9), 948-960.

Fong, S., Wong, R. ve Vasilakos, A. V., 2016, Accelerated PSO swarm search feature selection for data stream mining big data, *IEEE transactions on services computing*, 9 (1), 33-45.

Foster, I., 1995, Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering, *Addison Wesley*, New York.

Goldberg, D., 1989, Genetic Algorithms in Search, Optimization and Machine Learning, *Addison–Wesley Publishing Co*, Boston, MA.

Gonsalves, T. ve Egashira, A., 2013, Parallel swarms oriented particle swarm optimization, *Applied Computational Intelligence and Soft Computing*, 2013, 14.

Gould, N., 2006, An introduction to algorithms for continuous optimization, *Oxford University Computing Laboratory Notes*, Oxford.

Grama, A., 2003, Introduction to parallel computing, *Addison Wesley*, England.

Grosan, C., Abraham, A. ve Chis, M., 2006, Swarm intelligence in data mining, *Springer*, Berlin, Heidelberg.

Gülcü, Ş. ve Kodaz, H., 2015, A novel parallel multi-swarm algorithm based on comprehensive learning particle swarm optimization, *Engineering Applications of Artificial Intelligence*, 45, 33-45.

Gülcü, Ş., Mahi, M., Baykan, Ö. K. ve Kodaz, H., 2016, A parallel cooperative hybrid method based on ant colony optimization and 3-Opt algorithm for solving traveling salesman problem, *Soft Computing*.

Gündüz, M., Kıran, M. S. ve Özceylan, E., 2014, A hierarchic approach based on swarm intelligence to solve the traveling salesman problem, *Turkish Journal of Electrical Engineering & Computer Sciences*, 23 (1), 103-117.

Haklı, H. ve Uğuz, H., 2014, A novel particle swarm optimization algorithm with Levy flight, *Applied Soft Computing*, 23, 333-345.

Hao, Z.-F., Cai, R.-C. ve Huang, H., 2006, An adaptive parameter control strategy for ACO, *2006 International Conference on Machine Learning and Cybernetics*, 203-206.

Hasenjäger, M., Sendhoff, B., Sonoda, T. ve Arima, T., 2005, Three dimensional evolutionary aerodynamic design optimization with CMA-ES, *Proceedings of the 2005 conference on Genetic and evolutionary computation*, 2173-2180.

Hong, L., Zhong-hua, L. ve Xue-bin, C., 2010, The applications and trends of high performance computing in finance, *Ninth International Symposium on Distributed Computing and Applications to Business Engineering and Science (DCABES 2010)*, 193-197.

Jensi, R. ve Jiji, G. W., 2016, An enhanced particle swarm optimization with levy flight for global optimization, *Applied Soft Computing*, 43, 248-261.

Jia, H., 2015, A novel hybrid optimization algorithm and its application in solving complex problem, *International Journal of Hybrid Information Technology*, 8 (2), 1-10.

Jiang, J., Gao, J., Li, G., Wu, C. ve Pei, Z., 2014, Hierarchical Solving Method for Large Scale TSP Problems, *International Symposium on Neural Networks*, 252-261.

Johnson, D. S. ve McGeoch, L. A., 1997, The traveling salesman problem: A case study in local optimization, *Local search in combinatorial optimization*, 1, 215-310.

Johnson, R. A. ve Bhattacharyya, G. K., 2014, Statistics: principles and methods, *John Wiley & Sons*, New York.

Jun-man, K. ve Yi, Z., 2012, Application of an Improved Ant Colony Optimization on Generalized Traveling Salesman Problem, *Energy Procedia*, 17, 319-325.

Junqiang, W. ve Aijia, O., 2012, A hybrid algorithm of ACO and delete-cross method for TSP, *International Conference on Industrial Control and Electronics Engineering (ICICEE 2012)*, 1694-1696.

Kadah, Y. M., Abd-Elmoniem, K. Z. ve Farag, A. A., 2011, Parallel computation in medical imaging applications, *International journal of biomedical imaging*, 2011, 1-2.

Kampolis, I. C. ve Giannakoglou, K. C., 2009, Distributed evolutionary algorithms with hierarchical evaluation, *Engineering Optimization*, 41 (11), 1037-1049.

Karaboga, D. ve Basturk, B., 2007, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, *Journal of global optimization*, 39 (3), 459-471.

Kartal, M., 2014, Bilimsel araştırmalarda hipotez testleri: parametrik ve nonparametrik teknikler, *Nobel Akademik Yayıncılık*, Ankara.

Kendall, G. ve Su, Y., 2005, A Particle Swarm Optimisation Approach in the Construction of Optimal Risky Portfolios, *Artificial Intelligence and Applications*, 140-145.

Kennedy, J. ve Eberhart, R., 1995, Particle swarm optimization, *IEEE International Conference on Neural Networks Proceedings*, 1942-1948.

Kennedy, J. ve Mendes, R., 2002, Population structure and particle swarm performance, *Proceedings of the 2002 Congress on Evolutionary Computation CEC'02*, 1671-1676.

Kim, M., Lee, Y., Park, H.-H., Hahn, S. J. ve Lee, C.-G., 2015, Computational fluid dynamics simulation based on hadoop ecosystem and heterogeneous computing, *Computers & Fluids*, 115, 1-10.

Kıran, M. S., Özceylan, E., Gündüz, M. ve Paksoy, T., 2012, A novel hybrid approach based on particle swarm optimization and ant colony algorithm to forecast energy demand of Turkey, *Energy Conversion and Management*, 53 (1), 75-83.

Kızılateş, G., Nuriyeva, F. ve Kutucu, H., 2015, A tour extending hyper-heuristic algorithm for the traveling salesman problem, *Proceedings of IAM*, 4 (1), 8-15.

Kolli, V. M. ve Sharvani, G., 2014, An ACO-Based Efficient Stagnation Avoidance Methodology for MANETS, *Proceedings of Ninth International Conference on Wireless Communication and Sensor Networks*, 125-132.

Kotinis, M., 2011, Implementing co-evolution and parallelization in a multi-objective particle swarm optimizer, *Engineering Optimization*, 43 (6), 635-656.

Lee, A., Yau, C., Giles, M. B., Doucet, A. ve Holmes, C. C., 2010, On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods, *Journal of computational and graphical statistics*, 19 (4), 769-789.

Li, B. ve Wada, K., 2011, Communication latency tolerant parallel algorithm for particle swarm optimization, *Parallel Computing*, 37 (1), 1-10.

Li, X., 2010, Niching without niching parameters: particle swarm optimization using a ring topology, *IEEE Transactions on Evolutionary Computation*, 14 (1), 150-169.

Liang, J. J., Qin, A. K., Suganthan, P. N. ve Baskar, S., 2006, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, *IEEE Transactions on Evolutionary Computation*, 10 (3), 281-295.

Liu, J., Yang, J., Liu, H., Tian, X. ve Gao, M., 2016, An improved ant colony algorithm for robot path planning, *Soft Computing*, 1-11.

Llnl, 2016, Paralel Hesaplamaya Giriş, http://www.rc.unesp.br/igce/demac/balthazar/gpacp/bibliografia/A-05%20-%20Introduction%20to%20Parallel%20Computing%20-%20LLNL%20-%20Universit.htm: [05.12.2016].

Lucasius, C. B. ve Kateman, G., 1991, Genetic algorithms for large-scale optimization in chemometrics: an application, *TrAC Trends in Analytical Chemistry*, 10 (8), 254-261.

Mahi, M., Baykan, Ö. K. ve Kodaz, H., 2015, A new hybrid method based on Particle Swarm Optimization, Ant Colony Optimization and 3-Opt algorithms for Traveling Salesman Problem, *Applied Soft Computing*, 30, 484-490.

Marinakis, Y., Marinaki, M. ve Dounias, G., 2011, Honey bees mating optimization algorithm for the Euclidean traveling salesman problem, *Information Sciences*, 181 (20), 4684-4698.

Masutti, T. A. ve de Castro, L. N., 2009, A self-organizing neural network using ideas from the immune system to solve the traveling salesman problem, *Information Sciences*, 179 (10), 1454-1468.

Mendes, R., Kennedy, J. ve Neves, J., 2004, The fully informed particle swarm: simpler, maybe better, *IEEE Transactions on Evolutionary Computation*, 8 (3), 204-210.

Mirjalili, S., Mirjalili, S. M. ve Lewis, A., 2014, Grey wolf optimizer, *Advances in Engineering Software*, 69, 46-61.

Nalepa, J., Blocho, M. ve Czech, Z. J., 2013, Co-operation schemes for the parallel memetic algorithm, In: Parallel Processing and Applied Mathematics, Eds: Wyrzykowski, R., Dongarra, J., Karczewski, K. ve Waśniewski, J., *Springer*, Berlin, Heidelberg.

Neri, F., Cotta, C. ve Moscato, P., 2011, Handbook of memetic algorithms, *Springer*, Berlin, Heidelberg.

Norouzi, N., Sadegh-Amalnick, M. ve Tavakkoli-Moghaddam, R., 2017, Modified particle swarm optimization in a time-dependent vehicle routing problem: minimizing fuel consumption, *Optimization Letters*, 121-134.

Olhofer, M., Jin, Y. ve Sendhoff, B., 2001, Adaptive encoding for aerodynamic shape optimization using evolution strategies, *Proceedings of the 2001 Congress on Evolutionary Computation, 2001*, 576-583.

Omkar, S., Venkatesh, A. ve Mudigere, M., 2012, MPI-based parallel synchronous vector evaluated particle swarm optimization for multi-objective design optimization of composite structures, *Engineering Applications of Artificial Intelligence*, 25 (8), 1611-1627.

Orman, A. ve Williams, H. P., 2006, A survey of different integer programming formulations of the travelling salesman problem, *Optimisation, Econometric and Financial Analysis*, 9, 93-108.

Orobitg, M., Guirado, F., Cores, F., Llados, J. ve Notredame, C., 2015, High performance computing improvements on bioinformatics consistency-based multiple sequence alignment tools, *Parallel Computing*, 42, 18-34.

Othman, Z. A., Srour, A. I., Hamdan, A. R. ve Ling, P. Y., 2013, Performance Water Flow-Like Algorithm for TSP by improving its Local Search, *International Journal of Advancements in Computing Technology*, 5 (14), 126.

Parsopoulos, K. ve Vrahatis, M., 2005, Unified particle swarm optimization for tackling operations research problems, *Proceedings 2005 IEEE Swarm Intelligence Symposium (SIS 2005)*, 53-59.

Pasti, R. ve De Castro, L. N., 2006, A neuro-immune network for solving the traveling salesman problem, *International Joint Conference on Neural Networks (IJCNN 2006)*, 3760-3766.

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

Peker, M., Sen, B. ve Kumru, P. Y., 2013, An efficient solving of the traveling salesman problem: the ant colony system having parameters optimized by the Taguchi method, *Turkish Journal of Electrical Engineering & Computer Sciences*, 21 (Sup. 1), 2015-2036.

Peram, T., Veeramachaneni, K. ve Mohan, C. K., 2003, Fitness-distance-ratio based particle swarm optimization, *Proceedings of the 2003 IEEE Swarm Intelligence Symposium (SIS 2003)*, 174-181.

ProQuest, ProQuest periyodik kaynak veritabanı, http://search.proquest.com/central: [18.12.2016].

Qu, B.-Y., Suganthan, P. N. ve Das, S., 2013, A distance-based locally informed particle swarm model for multimodal optimization, *IEEE Transactions on Evolutionary Computation*, 17 (3), 387-402.

Ray, T. ve Smith, W., 2006, A surrogate assisted parallel multiobjective evolutionary algorithm for robust engineering design, *Engineering Optimization*, 38 (8), 997-1011.

Reinelt, G., 1991, TSPLIB—A traveling salesman problem library, *ORSA journal on computing*, 3 (4), 376-384.

Reinelt, G., 1995, TSP library, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/: [20.12.2016].

Romero-Laorden, D., Villazón-Terrazas, J., Martínez-Graullera, O., Ibáñez, A., Parrilla, M. ve Penas, M. S., 2016, Analysis of parallel computing strategies to accelerate ultrasound imaging processes, *IEEE Transactions on Parallel and Distributed Systems*, 27 (12), 3429-3440.

Ross, S. M., 2014, Introduction to probability and statistics for engineers and scientists, *Academic Press*, Boston.

Ruciński, M., Izzo, D. ve Biscani, F., 2010, On the impact of the migration topology on the island model, *Parallel Computing*, 36 (10), 555-571.

Salomon, R., 1996, Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. A survey of some theoretical and practical aspects of genetic algorithms, *BioSystems*, 39 (3), 263-278.

Savic, D., 2002, Single-objective vs. multiobjective optimisation for integrated decision support, *Integrated Assessment and Decision Support*, 1, 7-12.

Schäffler, S., 2012, Global Optimization: A Stochastic Approach, *Springer*, New York.

Scheck, R., 2008, Create Dynamic Charts in Microsoft Office Excel 2007 and Beyond, *Microsoft Press*, Washington.

Schwaab, M., Biscaia Jr, E. C., Monteiro, J. L. ve Pinto, J. C., 2008, Nonlinear parameter estimation through particle swarm optimization, *Chemical Engineering Science*, 63 (6), 1542-1552.

Shang, Y.-W. ve Qiu, Y.-H., 2006, A note on the extended Rosenbrock function, *Evolutionary Computation*, 14 (1), 119-126.

Shi, Y. ve Eberhart, R., 1998, A modified particle swarm optimizer, *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence*, 69-73.

Shi, Y. ve Eberhart, R. C., 1999, Empirical study of particle swarm optimization, *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99*.

Snyder, L. V. ve Daskin, M. S., 2006, A random-key genetic algorithm for the generalized traveling salesman problem, *European Journal of Operational Research*, 174 (1), 38-53.

Stützle, T., López-Ibánez, M., Pellegrini, P., Maur, M., De Oca, M. M., Birattari, M. ve Dorigo, M., 2011, Parameter adaptation in ant colony optimization, In: Autonomous search, Eds: Hamadi, Y., Monfroy, E. ve Saubion, F., *Springer*, Berlin, Heidelberg.

Talbi, E.-G., 2009, Metaheuristics: from design to implementation, *John Wiley & Sons*, New Jersey.

Tanenbaum, A. ve Van Steen, M., 2007, Distributed Systems: Principles and Paradigms, *Pearson Prentice Hall*, Upper Saddle River.

Tang, J., Lim, M.-H., Ong, Y.-S. ve Er, M. J., 2004, Study of migration topology in island model parallel hybrid-GA for large scale quadratic assignment problems, *8th International Conference on Control, Automation, Robotics and Vision (ICARCV 2004)*, 2286-2291.

Tang, K., Li, Z., Luo, L. ve Liu, B., 2015, Multi-strategy adaptive particle swarm optimization for numerical optimization, *Engineering Applications of Artificial Intelligence*, 37, 9-19.

Torn, A. ve Zilinskas, A., 1989, Global Optimization, *Lecture Notes in Computer Science*, 350, 1-252.

Tsai, C.-F., Tsai, C.-W. ve Tseng, C.-C., 2004, A new hybrid heuristic approach for solving large traveling salesman problem, *Information Sciences*, 166 (1), 67-81.

Tsai, C.-F., Lin, W.-C. ve Ke, S.-W., 2016, Big data mining with parallel computing: A comparison of distributed and MapReduce methodologies, *Journal of Systems and Software*, 122, 83-92.

Tsutsui, S. ve Collet, P., 2013, Massively Parallel Evolutionary Computation on GPGPUs, *Springer*, Berlin, Heidelberg.

Van den Bergh, F. ve Engelbrecht, A. P., 2004, A cooperative approach to particle swarm optimization, *IEEE Transactions on Evolutionary Computation*, 8 (3), 225-239.

Van Den Bergh, F., 2006, An analysis of particle swarm optimizers, Ph.D. Thesis, *University of Pretoria, the Faculty of Natural and Agricultural Science*, Pretoria.

Viswanathan, V. ve Krishnamurthi, I., 2015, Finding relevant semantic association paths using semantic ant colony optimization algorithm, *Soft Computing*, 19 (1), 251-260.

Wang, Y., 2014, The hybrid genetic algorithm with two local optimization strategies for traveling salesman problem, *Computers & Industrial Engineering*, 70, 124-133.

Wolpert, D. H. ve Macready, W. G., 1997, No free lunch theorems for optimization, *IEEE Transactions on Evolutionary Computation*, 1 (1), 67-82.

Wu, J. ve Tan, Y., 2009, A particle swarm optimization algorithm for grain logistics vehicle routing problem, *ISECS International Colloquium on Computing, Communication, Control, and Management* 364-367.

Wu, M.-T., Hong, T.-P. ve Lee, C.-N., 2012, A continuous ant colony system framework for fuzzy data mining, *Soft Computing*, 16 (12), 2071-2082.

Yang, X.-S., Deb, S. ve Fong, S., 2011, Accelerated particle swarm optimization and support vector machine for business optimization and applications, In: Networked digital technologies, Eds: Fong, S., *Springer*, Berlin, Heidelberg.

Yao, X., Liu, Y. ve Lin, G., 1999, Evolutionary programming made faster, *IEEE Transactions on Evolutionary Computation*, 3 (2), 82-102.

Yıldız, A., 2009, A novel particle swarm optimization approach for product design and manufacturing, *The International Journal of Advanced Manufacturing Technology*, 40 (5-6), 617-628.

Yildiz, A. R., 2012, A new hybrid particle swarm optimization approach for structural design optimization in the automotive industry, *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 226 (10), 1340-1351.

Yong, W., 2015, Hybrid Max–Min ant system with four vertices and three lines inequality for traveling salesman problem, *Soft Computing*, 19 (3), 585-596.

Yuan, F., 2015, Modeling and computational strategies for medical decision making, Ph.D. Thesis, *Georgia Institute of Technology*, Atlanta, Georgia.

Zaki, M. J. ve Ho, C.-T., 2000, Large-scale parallel data mining, 1759, *Springer*, Berlin, Heidelberg.

Zhang, G.-W., Zhan, Z.-H., Du, K.-J., Lin, Y., Chen, W.-N., Li, J.-J. ve Zhang, J., 2015, Parallel Particle Swarm Optimization Using Message Passing Interface, *Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems, Volume 1*, 55-64.

Zhang, K.-n., Wu, Y., Ding, C., Pruess, K. ve Elmroth, E., 2001, Parallel computing techniques for large-scale reservoir simulation of multi-component and multiphase fluid flow, *SPE Reservoir Simulation Symposium*.

Zhou, Y., Luo, Q., Chen, H., He, A. ve Wu, J., 2015, A discrete invasive weed optimization algorithm for solving traveling salesman problem, *Neurocomputing*, 151, 1227-1236.

Zhu, Y.-M. ve Cochoff, S. M., 2010, Medical image viewing on multicore platforms using parallel computing patterns, *IT professional*, 12 (2), 33-41.

Zomaya, A. Y., 2006, Parallel computing for bioinformatics and computational biology: models, enabling technologies, and case studies, *John Wiley & Sons*, Hoboken, New Jersey.

# EKLER

**EK-1** Wilcoxon sıra toplam testine ait çizelgeler.

**Çizelge EK-1.1.** Wilcoxon sıra toplam testi kritik değerler tablosu ($\alpha$ = 0.025 tek yönlü, $\alpha$ = 0.05 çift yönlü) (Kartal, 2014)

| # | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | | 9 | | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $T_L$ | $T_U$ | $T_L$ | $T_U$ | $T_L$ | $T_U$ | $T_L$ | $T_U$ | $T_L$ | $T_U$ | $T_L$ | $T_U$ | $T_L$ | $T_U$ | $T_L$ | $T_U$ |
| 3 | 5 | 16 | 6 | 18 | 6 | 21 | 7 | 23 | 7 | 26 | 8 | 28 | 8 | 31 | 9 | 33 |
| 4 | 6 | 18 | 11 | 25 | 12 | 28 | 12 | 32 | 13 | 35 | 14 | 38 | 15 | 41 | 16 | 44 |
| 5 | 6 | 21 | 12 | 28 | 18 | 37 | 19 | 41 | 20 | 45 | 21 | 49 | 22 | 53 | 24 | 56 |
| 6 | 7 | 23 | 12 | 32 | 19 | 41 | 26 | 52 | 28 | 56 | 29 | 61 | 31 | 65 | 32 | 70 |
| 7 | 7 | 26 | 13 | 35 | 20 | 45 | 28 | 56 | 37 | 68 | 39 | 73 | 41 | 78 | 43 | 83 |
| 8 | 8 | 28 | 14 | 38 | 21 | 49 | 29 | 61 | 39 | 73 | 49 | 87 | 51 | 93 | 54 | 98 |
| 9 | 8 | 31 | 15 | 41 | 22 | 53 | 31 | 65 | 41 | 78 | 51 | 93 | 63 | 108 | 66 | 114 |
| 10 | 9 | 33 | 16 | 44 | 24 | 56 | 32 | 70 | 43 | 83 | 54 | 98 | 66 | 114 | 79 | 131 |

**Çizelge EK-1.2.** Wilcoxon sıra toplam testi kritik değerler tablosu ($\alpha$ = 0.05 tek yönlü, $\alpha$ = 0.10 çift yönlü) (Kartal, 2014)

| # | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | | 9 | | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $T_L$ | $T_U$ | $T_L$ | $T_U$ | $T_L$ | $T_U$ | $T_L$ | $T_U$ | $T_L$ | $T_U$ | $T_L$ | $T_U$ | $T_L$ | $T_U$ | $T_L$ | $T_U$ |
| 3 | 6 | 15 | 7 | 17 | 7 | 20 | 8 | 22 | 9 | 24 | 9 | 27 | 10 | 29 | 11 | 31 |
| 4 | 7 | 17 | 12 | 24 | 13 | 27 | 14 | 30 | 15 | 33 | 16 | 36 | 17 | 39 | 18 | 42 |
| 5 | 7 | 20 | 13 | 27 | 19 | 36 | 20 | 40 | 22 | 43 | 24 | 46 | 25 | 50 | 26 | 54 |
| 6 | 8 | 22 | 14 | 30 | 20 | 40 | 28 | 50 | 30 | 54 | 32 | 58 | 33 | 63 | 35 | 67 |
| 7 | 9 | 24 | 15 | 33 | 22 | 43 | 30 | 54 | 39 | 66 | 41 | 71 | 43 | 76 | 46 | 80 |
| 8 | 9 | 27 | 16 | 36 | 24 | 46 | 32 | 58 | 41 | 71 | 52 | 84 | 54 | 90 | 57 | 95 |
| 9 | 10 | 29 | 17 | 39 | 25 | 50 | 33 | 63 | 43 | 76 | 54 | 90 | 66 | 105 | 69 | 111 |
| 10 | 11 | 31 | 18 | 42 | 26 | 54 | 35 | 67 | 46 | 80 | 57 | 95 | 69 | 111 | 83 | 127 |

**Çizelge EK-1.3.** Standart normal eğri alanları (Z tablosu) (Kartal, 2014)

| Z | 0.00 | 0.01 | 0.02 | 0.03 | 0.04 | 0.05 | 0.06 | 0.07 | 0.08 | 0.09 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0.0 | 0.0000 | 0.0040 | 0.0080 | 0.0120 | 0.0160 | 0.0199 | 0.0239 | 0.0279 | 0.0319 | 0.0359 |
| 0.1 | 0.0398 | 0.0438 | 0.0478 | 0.0517 | 0.0557 | 0.0596 | 0.0636 | 0.0675 | 0.0714 | 0.0753 |
| 0.2 | 0.0793 | 0.0832 | 0.0871 | 0.0910 | 0.0948 | 0.0987 | 0.1026 | 0.1064 | 0.1103 | 0.1141 |
| 0.3 | 0.1179 | 0.1217 | 0.1255 | 0.1293 | 0.1331 | 0.1368 | 0.1406 | 0.1443 | 0.1480 | 0.1517 |
| 0.4 | 0.1554 | 0.1591 | 0.1628 | 0.1664 | 0.1700 | 0.1736 | 0.1772 | 0.1808 | 0.1844 | 0.1879 |
| 0.5 | 0.1915 | 0.1950 | 0.1985 | 0.2019 | 0.2054 | 0.2088 | 0.2123 | 0.2157 | 0.2190 | 0.2224 |
| 0.6 | 0.2257 | 0.2291 | 0.2324 | 0.2357 | 0.2389 | 0.2422 | 0.2454 | 0.2486 | 0.2517 | 0.2549 |
| 0.7 | 0.2580 | 0.2611 | 0.2642 | 0.2673 | 0.2704 | 0.2734 | 0.2764 | 0.2794 | 0.2823 | 0.2852 |
| 0.8 | 0.2881 | 0.2910 | 0.2939 | 0.2967 | 0.2995 | 0.3023 | 0.3051 | 0.3078 | 0.3106 | 0.3133 |
| 0.9 | 0.3159 | 0.3186 | 0.3212 | 0.3238 | 0.3264 | 0.3289 | 0.3315 | 0.3340 | 0.3365 | 0.3389 |
| 1.0 | 0.3413 | 0.3438 | 0.3461 | 0.3485 | 0.3508 | 0.3531 | 0.3554 | 0.3577 | 0.3599 | 0.3621 |
| 1.1 | 0.3643 | 0.3665 | 0.3686 | 0.3708 | 0.3729 | 0.3749 | 0.3770 | 0.3790 | 0.3810 | 0.3830 |
| 1.2 | 0.3849 | 0.3869 | 0.3888 | 0.3907 | 0.3925 | 0.3944 | 0.3962 | 0.3980 | 0.3997 | 0.4015 |
| 1.3 | 0.4032 | 0.4049 | 0.4066 | 0.4082 | 0.4099 | 0.4115 | 0.4131 | 0.4147 | 0.4162 | 0.4177 |
| 1.4 | 0.4192 | 0.4207 | 0.4222 | 0.4236 | 0.4251 | 0.4265 | 0.4279 | 0.4292 | 0.4306 | 0.4319 |
| 1.5 | 0.4332 | 0.4345 | 0.4357 | 0.4370 | 0.4382 | 0.4394 | 0.4406 | 0.4418 | 0.4429 | 0.4441 |
| 1.6 | 0.4452 | 0.4463 | 0.4474 | 0.4484 | 0.4495 | 0.4505 | 0.4515 | 0.4525 | 0.4535 | 0.4545 |
| 1.7 | 0.4554 | 0.4564 | 0.4573 | 0.4582 | 0.4591 | 0.4599 | 0.4608 | 0.4616 | 0.4625 | 0.4633 |
| 1.8 | 0.4641 | 0.4649 | 0.4656 | 0.4664 | 0.4671 | 0.4678 | 0.4686 | 0.4693 | 0.4699 | 0.4706 |
| 1.9 | 0.4713 | 0.4719 | 0.4726 | 0.4732 | 0.4738 | 0.4744 | 0.4750 | 0.4756 | 0.4761 | 0.4767 |
| 2.0 | 0.4772 | 0.4778 | 0.4783 | 0.4788 | 0.4793 | 0.4798 | 0.4803 | 0.4808 | 0.4812 | 0.4817 |
| 2.1 | 0.4821 | 0.4826 | 0.4830 | 0.4834 | 0.4838 | 0.4842 | 0.4846 | 0.4850 | 0.4854 | 0.4857 |
| 2.2 | 0.4861 | 0.4864 | 0.4868 | 0.4871 | 0.4875 | 0.4878 | 0.4881 | 0.4884 | 0.4887 | 0.4890 |
| 2.3 | 0.4893 | 0.4896 | 0.4898 | 0.4901 | 0.4904 | 0.4906 | 0.4909 | 0.4911 | 0.4913 | 0.4916 |
| 2.4 | 0.4918 | 0.4920 | 0.4922 | 0.4925 | 0.4927 | 0.4929 | 0.4931 | 0.4932 | 0.4934 | 0.4936 |
| 2.5 | 0.4938 | 0.4940 | 0.4941 | 0.4943 | 0.4945 | 0.4946 | 0.4948 | 0.4949 | 0.4951 | 0.4952 |
| 2.6 | 0.4953 | 0.4955 | 0.4956 | 0.4957 | 0.4959 | 0.4960 | 0.4961 | 0.4962 | 0.4963 | 0.4964 |
| 2.7 | 0.4965 | 0.4966 | 0.4967 | 0.4968 | 0.4969 | 0.4970 | 0.4971 | 0.4972 | 0.4973 | 0.4974 |
| 2.8 | 0.4974 | 0.4975 | 0.4976 | 0.4977 | 0.4977 | 0.4978 | 0.4979 | 0.4979 | 0.4980 | 0.4981 |
| 2.9 | 0.4981 | 0.4982 | 0.4982 | 0.4983 | 0.4984 | 0.4984 | 0.4985 | 0.4985 | 0.4986 | 0.4986 |
| 3.0 | 0.4986 | 0.4987 | 0.4987 | 0.4988 | 0.4988 | 0.4989 | 0.4989 | 0.4989 | 0.4990 | 0.4990 |
| 3.1 | 0.4990 | 0.4991 | 0.4991 | 0.4991 | 0.4992 | 0.4992 | 0.4992 | 0.4992 | 0.4993 | 0.4993 |
| 3.2 | 0.4993 | 0.4993 | 0.4994 | 0.4994 | 0.4994 | 0.4994 | 0.4994 | 0.4995 | 0.4995 | 0.4995 |
| 3.3 | 0.4995 | 0.4995 | 0.4995 | 0.4996 | 0.4996 | 0.4996 | 0.4996 | 0.4996 | 0.4996 | 0.4997 |
| 3.4 | 0.4997 | 0.4997 | 0.4997 | 0.4997 | 0.4997 | 0.4997 | 0.4997 | 0.4997 | 0.4997 | 0.4998 |
| 3.6 | 0.4998 | 0.4998 | 0.4999 | 0.4999 | 0.4999 | 0.4999 | 0.4999 | 0.4999 | 0.4999 | 0.4999 |
| 3.9 | 0.5000 | | | | | | | | | |

**Çizelge EK-1.4.** $f_1$ - $f_{14}$ için Wilcoxon sıra toplam testi sonuçları

| Sphere ($f_1$) | | | | |
|---|---|---|---|---|
| # | CLPSO | PCLPSO | Sıra CLPSO | Sıra PCLPSO |
| 1 | 1.38E-04 | 1.03E-08 | 44 | 29 |
| 2 | 1.80E-04 | 9.61E-09 | 58 | 27 |
| 3 | 1.62E-04 | 3.21E-09 | 54 | 11 |
| 4 | 1.14E-04 | 3.22E-09 | 36 | 12 |
| 5 | 1.49E-04 | 5.70E-09 | 51 | 17 |
| 6 | 1.63E-04 | 1.34E-10 | 55 | 8 |
| 7 | 8.25E-05 | 1.33E-10 | 31 | 7 |
| 8 | 1.18E-04 | 6.59E-09 | 38 | 19 |
| 9 | 1.64E-04 | 1.31E-10 | 56 | 6 |
| 10 | 1.10E-04 | 7.01E-09 | 35 | 21 |
| 11 | 1.37E-04 | 6.89E-09 | 43 | 20 |
| 12 | 1.45E-04 | 7.60E-09 | 47 | 22 |
| 13 | 1.30E-04 | 1.11E-10 | 40 | 5 |
| 14 | 1.50E-04 | 2.78E-09 | 50 | 10 |
| 15 | 1.15E-04 | 3.37E-09 | 37 | 13 |
| 16 | 1.90E-04 | 7.67E-09 | 60 | 23 |
| 17 | 1.43E-04 | 3.46E-09 | 46 | 14 |
| 18 | 1.60E-04 | 7.71E-09 | 53 | 24 |
| 19 | 1.59E-04 | 1.17E-08 | 52 | 30 |
| 20 | 1.42E-04 | 4.11E-09 | 45 | 15 |
| 21 | 1.48E-04 | 4.56E-09 | 49 | 16 |
| 22 | 1.35E-04 | 9.22E-09 | 41 | 26 |
| 23 | 1.23E-04 | 8.97E-11 | 39 | 4 |
| 24 | 1.83E-04 | 8.64E-11 | 59 | 3 |
| 25 | 1.71E-04 | 5.92E-09 | 57 | 18 |
| 26 | 1.46E-04 | 9.82E-09 | 48 | 28 |
| 27 | 9.57E-05 | 8.03E-11 | 33 | 2 |
| 28 | 9.01E-05 | 2.31E-09 | 32 | 9 |
| 29 | 1.02E-04 | 7.22E-11 | 34 | 1 |
| 30 | 1.36E-04 | 8.72E-09 | 42 | 25 |

| Rosenbrock ($f_2$) | | | |
|---|---|---|---|
| CLPSO | PCLPSO | Sıra CLPSO | Sıra PCLPSO |
| 9.42E+01 | 7.48E+01 | 36 | 28 |
| 9.44E+01 | 7.41E+01 | 39.5 | 27 |
| 9.62E+01 | 7.36E+01 | 56 | 26 |
| 9.33E+01 | 7.35E+01 | 34 | 25 |
| 9.72E+01 | 7.34E+01 | 60 | 24 |
| 9.47E+01 | 7.33E+01 | 42 | 23 |
| 9.50E+01 | 7.30E+01 | 45.5 | 22 |
| 9.52E+01 | 7.29E+01 | 47 | 20 |
| 9.58E+01 | 7.29E+01 | 53 | 20 |
| 9.53E+01 | 7.29E+01 | 49.5 | 20 |
| 9.43E+01 | 7.28E+01 | 37 | 18 |
| 9.62E+01 | 7.27E+01 | 56 | 16.5 |
| 9.10E+01 | 7.27E+01 | 30 | 16.5 |
| 9.46E+01 | 7.23E+01 | 39.5 | 15 |
| 9.54E+01 | 7.17E+01 | 49.5 | 14 |
| 9.68E+01 | 7.16E+01 | 59 | 11.5 |
| 9.50E+01 | 7.16E+01 | 45.5 | 11.5 |
| 9.24E+01 | 9.49E+01 | 32 | 44 |
| 9.46E+01 | 7.15E+01 | 39.5 | 11.5 |
| 9.61E+01 | 7.15E+01 | 54 | 11.5 |
| 9.30E+01 | 7.14E+01 | 33 | 9 |
| 9.53E+01 | 7.12E+01 | 49.5 | 8 |
| 9.44E+01 | 6.98E+01 | 39.5 | 7 |
| 9.66E+01 | 6.96E+01 | 58 | 4.5 |
| 9.62E+01 | 6.96E+01 | 56 | 4.5 |
| 9.48E+01 | 6.96E+01 | 43 | 4.5 |
| 9.13E+01 | 6.96E+01 | 31 | 4.5 |
| 9.55E+01 | 6.95E+01 | 52 | 2 |
| 9.38E+01 | 6.92E+01 | 35 | 1 |
| 9.54E+01 | 7.93E+01 | 49.5 | 29 |

| Ackley ($f_3$) | | | | |
|---|---|---|---|---|
| # | CLPSO | PCLPSO | Sıra CLPSO | Sıra PCLPSO |
| 1 | 2.63E-03 | 6.68E-06 | 41 | 3 |
| 2 | 2.69E-03 | 1.07E-05 | 46 | 12 |
| 3 | 3.13E-03 | 1.22E-05 | 54 | 16 |
| 4 | 2.88E-03 | 1.33E-05 | 49.5 | 23.5 |
| 5 | 3.17E-03 | 1.53E-05 | 56 | 29 |
| 6 | 3.16E-03 | 1.26E-05 | 55 | 18 |
| 7 | 2.64E-03 | 1.33E-05 | 42 | 23.5 |
| 8 | 3.42E-03 | 1.34E-06 | 58 | 2 |
| 9 | 2.33E-03 | 1.36E-05 | 34 | 25 |
| 10 | 2.32E-03 | 6.83E-06 | 33 | 4 |
| 11 | 2.68E-03 | 1.26E-05 | 45 | 18 |
| 12 | 2.98E-03 | 1.17E-05 | 51 | 15 |
| 13 | 2.56E-03 | 1.29E-06 | 39 | 1 |
| 14 | 2.88E-03 | 1.58E-05 | 49.5 | 30 |
| 15 | 3.06E-03 | 9.13E-06 | 52 | 9 |
| 16 | 2.10E-03 | 1.49E-05 | 31 | 28 |
| 17 | 3.08E-03 | 1.30E-05 | 53 | 20 |
| 18 | 2.49E-03 | 9.37E-06 | 36.5 | 10 |
| 19 | 3.21E-03 | 9.82E-06 | 57 | 11 |
| 20 | 2.59E-03 | 1.41E-05 | 40 | 26 |
| 21 | 3.59E-03 | 7.69E-06 | 60 | 7 |
| 22 | 2.67E-03 | 1.31E-05 | 44 | 21 |
| 23 | 2.66E-03 | 1.08E-05 | 43 | 13 |
| 24 | 2.49E-03 | 1.32E-05 | 36.5 | 22 |
| 25 | 2.42E-03 | 8.92E-06 | 35 | 8 |
| 26 | 2.75E-03 | 1.26E-05 | 48 | 18 |
| 27 | 2.52E-03 | 7.44E-06 | 38 | 6 |
| 28 | 3.53E-03 | 6.90E-06 | 59 | 5 |
| 29 | 2.15E-03 | 1.42E-05 | 32 | 27 |
| 30 | 2.73E-03 | 1.12E-05 | 47 | 14 |

| Griewank ($f_4$) | | | |
|---|---|---|---|
| CLPSO | PCLPSO | Sıra CLPSO | Sıra PCLPSO |
| 1.13E-04 | 1.72E-10 | 41 | 1 |
| 1.43E-04 | 1.22E-08 | 51 | 9 |
| 1.27E-04 | 2.94E-08 | 48 | 18 |
| 1.22E-04 | 2.58E-08 | 45.5 | 15 |
| 1.02E-04 | 1.39E-08 | 38 | 11 |
| 1.79E-04 | 6.60E-08 | 55 | 21 |
| 8.16E-05 | 9.03E-09 | 32 | 8 |
| 2.28E-04 | 5.55E-09 | 59 | 3 |
| 9.30E-05 | 6.76E-09 | 35 | 5 |
| 1.32E-04 | 1.81E-08 | 49 | 12 |
| 2.97E-04 | 2.70E-08 | 60 | 16 |
| 1.96E-04 | 1.17E-07 | 56 | 25 |
| 1.21E-04 | 5.36E-08 | 44 | 20 |
| 1.08E-04 | 5.59E-09 | 39 | 4 |
| 8.39E-05 | 3.93E-07 | 33 | 28 |
| 2.00E-04 | 4.02E-07 | 57 | 29 |
| 2.09E-04 | 9.80E-08 | 58 | 24 |
| 1.14E-04 | 8.40E-08 | 42 | 23 |
| 1.54E-04 | 8.27E-08 | 53 | 22 |
| 1.41E-04 | 5.15E-09 | 50 | 2 |
| 8.95E-05 | 2.45E-08 | 34 | 14 |
| 1.46E-04 | 1.30E-08 | 52 | 10 |
| 7.47E-05 | 1.48E-07 | 31 | 26 |
| 1.23E-04 | 3.44E-08 | 47 | 19 |
| 1.22E-04 | 6.77E-09 | 45.5 | 6 |
| 9.34E-05 | 1.87E-08 | 36 | 13 |
| 1.09E-04 | 3.29E-06 | 40 | 30 |
| 1.71E-04 | 2.79E-08 | 54 | 17 |
| 1.18E-04 | 8.29E-09 | 43 | 7 |
| 9.89E-05 | 1.64E-07 | 37 | 27 |

| Weierstrass ($f_5$) | | | | |
|---|---|---|---|---|
| # | CLPSO | PCLPSO | Sıra CLPSO | Sıra PCLPSO |
| 1 | 6.71E+00 | 5.96E-04 | 58 | 4 |
| 2 | 2.48E+00 | 0.001194 | 31 | 29 |
| 3 | 3.73E+00 | 1.62E-04 | 35 | 3 |
| 4 | 6.47E+00 | 6.73E-04 | 56 | 6 |
| 5 | 3.71E+00 | 9.17E-04 | 34 | 23 |
| 6 | 6.26E+00 | 7.02E-04 | 54 | 9 |
| 7 | 3.90E+00 | 1.57E-04 | 36 | 2 |
| 8 | 4.61E+00 | 7.82E-04 | 41 | 19 |
| 9 | 5.19E+00 | 7.19E-04 | 49 | 11 |
| 10 | 6.35E+00 | 0.001067 | 55 | 28 |
| 11 | 4.81E+00 | 0.001214 | 45 | 30 |
| 12 | 3.99E+00 | 7.02E-04 | 37 | 9 |
| 13 | 4.22E+00 | 9.05E-04 | 39 | 22 |
| 14 | 5.06E+00 | 9.71E-04 | 47 | 26 |
| 15 | 3.31E+00 | 7.41E-04 | 32 | 12 |
| 16 | 5.72E+00 | 9.59E-04 | 51 | 25 |
| 17 | 4.03E+00 | 1.48E-04 | 38 | 1 |
| 18 | 5.90E+00 | 7.02E-04 | 53 | 9 |
| 19 | 4.71E+00 | 7.42E-04 | 42 | 13 |
| 20 | 5.75E+00 | 8.67E-04 | 52 | 21 |
| 21 | 5.22E+00 | 7.60E-04 | 50 | 16 |
| 22 | 5.05E+00 | 6.78E-04 | 46 | 7 |
| 23 | 4.78E+00 | 7.47E-04 | 43 | 14 |
| 24 | 4.80E+00 | 6.08E-04 | 44 | 5 |
| 25 | 4.54E+00 | 0.001009 | 40 | 27 |
| 26 | 7.94E+00 | 8.29E-04 | 60 | 20 |
| 27 | 6.59E+00 | 7.65E-04 | 57 | 17 |
| 28 | 3.37E+00 | 7.57E-04 | 33 | 15 |
| 29 | 5.16E+00 | 9.33E-04 | 48 | 24 |
| 30 | 6.92E+00 | 7.77E-04 | 59 | 18 |

| Rastrigin ($f_6$) | | | |
|---|---|---|---|
| CLPSO | PCLPSO | Sıra CLPSO | Sıra PCLPSO |
| 1.34E+01 | 1.11E-03 | 38 | 21 |
| 1.07E+01 | 3.24E-04 | 31 | 17 |
| 1.75E+01 | 4.80E-03 | 53 | 29 |
| 1.25E+01 | 1.56E-04 | 34 | 11 |
| 1.35E+01 | 3.34E-04 | 39 | 18 |
| 2.02E+01 | 3.26E-05 | 57 | 2 |
| 1.98E+01 | 1.23E-04 | 56 | 7.5 |
| 2.07E+01 | 8.07E-03 | 58 | 30 |
| 1.60E+01 | 8.10E-05 | 46 | 5 |
| 2.14E+01 | 1.52E-04 | 59 | 10 |
| 1.65E+01 | 7.26E-04 | 48 | 20 |
| 1.54E+01 | 6.67E-05 | 43 | 4 |
| 1.45E+01 | 2.51E-04 | 41 | 15 |
| 1.49E+01 | 1.50E-03 | 42 | 25 |
| 1.28E+01 | 1.40E-04 | 36 | 9 |
| 1.64E+01 | 8.45E-05 | 47 | 6 |
| 1.20E+01 | 1.23E-04 | 33 | 7.5 |
| 1.74E+01 | 1.61E-04 | 52 | 12 |
| 1.69E+01 | 1.77E-03 | 50 | 26 |
| 1.57E+01 | 2.15E-04 | 44.5 | 14 |
| 1.97E+01 | 2.47E-05 | 55 | 1 |
| 1.27E+01 | 3.05E-04 | 35 | 16 |
| 1.19E+01 | 1.63E-04 | 32 | 13 |
| 1.94E+01 | 1.48E-03 | 54 | 24 |
| 1.29E+01 | 3.36E-03 | 37 | 28 |
| 1.72E+01 | 5.03E-04 | 51 | 19 |
| 1.40E+01 | 3.27E-05 | 40 | 3 |
| 2.15E+01 | 1.45E-03 | 60 | 23 |
| 1.67E+01 | 2.84E-03 | 49 | 27 |
| 1.57E+01 | 1.21E-03 | 44.5 | 22 |

| Non Cont. Rastrigin ($f_7$) | | | | |
|---|---|---|---|---|
| # | CLPSO | PCLPSO | Sıra CLPSO | Sıra PCLPSO |
| 1 | 1.28E+00 | 5.94E+01 | 20 | 60 |
| 2 | 1.32E+00 | 3.65E+01 | 23 | 59 |
| 3 | 3.26E+00 | 1.67E+01 | 39 | 49 |
| 4 | 2.51E+00 | 8.31E+00 | 36.5 | 45 |
| 5 | 4.19E+00 | 2.17E+00 | 41 | 31 |
| 6 | 1.27E+00 | 1.91E+00 | 18 | 29 |
| 7 | 1.28E+00 | 1.80E+00 | 20 | 28 |
| 8 | 1.16E+00 | 7.15E-01 | 14 | 12 |
| 9 | 2.35E+00 | 2.32E+01 | 35 | 55 |
| 10 | 1.70E+00 | 6.57E-01 | 26 | 11 |
| 11 | 1.17E+00 | 6.07E-01 | 15 | 10 |
| 12 | 2.99E-01 | 1.88E+01 | 7.5 | 51 |
| 13 | 1.15E+00 | 2.88E+01 | 13 | 57 |
| 14 | 5.16E+00 | 3.37E-01 | 44 | 9 |
| 15 | 5.14E+00 | 1.59E+01 | 43 | 48 |
| 16 | 2.51E+00 | 2.88E-01 | 36.5 | 6 |
| 17 | 1.18E+00 | 2.26E+00 | 16 | 34 |
| 18 | 4.27E+00 | 3.21E+01 | 42 | 58 |
| 19 | 1.28E+00 | 9.42E+00 | 20 | 46 |
| 20 | 2.23E+00 | 1.78E+00 | 33 | 27 |
| 21 | 2.99E-01 | 1.91E+01 | 7.5 | 53 |
| 22 | 2.43E-01 | 1.68E+00 | 4 | 25 |
| 23 | 2.22E+00 | 1.72E+01 | 32 | 50 |
| 24 | 2.15E+00 | 1.89E+01 | 30 | 52 |
| 25 | 1.26E+00 | 2.39E+01 | 17 | 56 |
| 26 | 1.51E+00 | 2.08E+01 | 24 | 54 |
| 27 | 3.52E+00 | 1.36E-01 | 40 | 3 |
| 28 | 1.31E+00 | 1.26E-01 | 22 | 2 |
| 29 | 3.21E+00 | 1.23E-01 | 38 | 1 |
| 30 | 2.66E-01 | 1.18E+01 | 5 | 47 |

| Schwefel ($f_8$) | | | |
|---|---|---|---|
| CLPSO | PCLPSO | Sıra CLPSO | Sıra PCLPSO |
| 4.15E-03 | 1.27E-03 | 38 | 15.5 |
| 4.57E-03 | 1.27E-03 | 40 | 15.5 |
| 5.24E-03 | 1.27E-03 | 47 | 15.5 |
| 1.18E+02 | 1.27E-03 | 55.5 | 15.5 |
| 4.59E-03 | 1.27E-03 | 41 | 15.5 |
| 3.36E-03 | 1.27E-03 | 33 | 15.5 |
| 1.18E+02 | 1.27E-03 | 55.5 | 15.5 |
| 5.32E-03 | 1.27E-03 | 48 | 15.5 |
| 7.14E-03 | 1.27E-03 | 52 | 15.5 |
| 5.46E-03 | 1.27E-03 | 49 | 15.5 |
| 1.18E+02 | 1.27E-03 | 55.5 | 15.5 |
| 3.62E-03 | 1.27E-03 | 35 | 15.5 |
| 3.00E-03 | 1.27E-03 | 32 | 15.5 |
| 4.83E-03 | 1.27E-03 | 42 | 15.5 |
| 6.45E-03 | 1.27E-03 | 51 | 15.5 |
| 3.85E-03 | 1.27E-03 | 36 | 15.5 |
| 3.99E-03 | 1.27E-03 | 37 | 15.5 |
| 5.02E-03 | 1.27E-03 | 44.5 | 15.5 |
| 6.22E-03 | 1.27E-03 | 50 | 15.5 |
| 2.37E+02 | 1.27E-03 | 59.5 | 15.5 |
| 1.18E+02 | 1.27E-03 | 55.5 | 15.5 |
| 4.84E-03 | 1.27E-03 | 44.5 | 15.5 |
| 2.70E-03 | 1.27E-03 | 31 | 15.5 |
| 3.48E-03 | 1.27E-03 | 34 | 15.5 |
| 2.37E+02 | 1.27E-03 | 59.5 | 15.5 |
| 1.18E+02 | 1.27E-03 | 55.5 | 15.5 |
| 4.84E-03 | 1.27E-03 | 44.5 | 15.5 |
| 4.53E-03 | 1.27E-03 | 39 | 15.5 |
| 5.02E-03 | 1.27E-03 | 44.5 | 15.5 |
| 1.18E+02 | 1.27E-03 | 55.5 | 15.5 |

| Rotated Ackley ($f_9$) | | | | |
|---|---|---|---|---|
| # | CLPSO | PCLPSO | Sıra CLPSO | Sıra PCLPSO |
| 1 | 2.97E-03 | 1.92E-05 | 52 | 10 |
| 2 | 2.74E-03 | 1.69E-05 | 45 | 8 |
| 3 | 2.49E-03 | 2.60E-06 | 39.5 | 5 |
| 4 | 2.78E-03 | 2.59E-06 | 46 | 4 |
| 5 | 3.06E-03 | 3.16E-05 | 56 | 25 |
| 6 | 1.99E-03 | 2.50E-06 | 31 | 3 |
| 7 | 2.93E-03 | 2.18E-05 | 51 | 12 |
| 8 | 2.30E-03 | 3.85E-05 | 35 | 29 |
| 9 | 2.66E-03 | 2.98E-05 | 42 | 23 |
| 10 | 3.25E-03 | 2.71E-05 | 59 | 20 |
| 11 | 2.98E-03 | 2.42E-06 | 53 | 2 |
| 12 | 3.11E-03 | 2.35E-06 | 57 | 1 |
| 13 | 2.83E-03 | 3.20E-05 | 48 | 26 |
| 14 | 2.20E-03 | 2.39E-05 | 34 | 16 |
| 15 | 2.35E-03 | 2.22E-05 | 36 | 13 |
| 16 | 2.42E-03 | 1.51E-05 | 37 | 7 |
| 17 | 3.19E-03 | 1.86E-05 | 58 | 9 |
| 18 | 2.79E-03 | 2.50E-05 | 47 | 18 |
| 19 | 2.46E-03 | 2.45E-05 | 38 | 17 |
| 20 | 2.19E-03 | 2.03E-05 | 33 | 11 |
| 21 | 2.73E-03 | 4.34E-05 | 44 | 30 |
| 22 | 3.37E-03 | 2.27E-05 | 60 | 14 |
| 23 | 3.00E-03 | 1.39E-05 | 54 | 6 |
| 24 | 2.89E-03 | 2.83E-05 | 50 | 22 |
| 25 | 2.07E-03 | 2.79E-05 | 32 | 21 |
| 26 | 2.49E-03 | 3.53E-05 | 39.5 | 28 |
| 27 | 2.58E-03 | 3.09E-05 | 41 | 24 |
| 28 | 2.72E-03 | 2.28E-05 | 43 | 15 |
| 29 | 2.84E-03 | 2.69E-05 | 49 | 19 |
| 30 | 3.04E-03 | 3.48E-05 | 55 | 27 |

| Rotated Griewank ($f_{10}$) | | | |
|---|---|---|---|
| CLPSO | PCLPSO | Sıra CLPSO | Sıra PCLPSO |
| 3.53E-03 | 1.53E-05 | 46 | 19 |
| 3.86E-03 | 3.77E-06 | 48 | 10 |
| 4.36E-03 | 9.85E-05 | 54 | 27 |
| 1.98E-03 | 7.94E-04 | 38 | 30 |
| 5.83E-03 | 5.46E-06 | 58 | 12 |
| 1.62E-03 | 2.55E-06 | 34 | 5 |
| 8.58E-04 | 7.14E-06 | 31 | 13 |
| 5.63E-03 | 1.45E-05 | 57 | 18 |
| 2.33E-03 | 1.15E-05 | 39 | 17 |
| 1.33E-03 | 3.03E-05 | 33 | 21 |
| 2.68E-03 | 5.58E-05 | 40 | 23 |
| 4.35E-03 | 9.91E-07 | 53 | 4 |
| 4.46E-03 | 6.84E-05 | 55 | 24 |
| 2.79E-03 | 2.67E-06 | 41 | 6 |
| 3.82E-03 | 3.69E-06 | 47 | 9 |
| 1.69E-03 | 2.03E-05 | 35 | 20 |
| 4.00E-03 | 7.32E-05 | 50 | 25 |
| 1.17E-03 | 8.15E-06 | 32 | 14 |
| 3.14E-03 | 9.27E-06 | 43.5 | 15 |
| 1.70E-03 | 1.07E-05 | 36 | 16 |
| 3.91E-03 | 1.04E-04 | 49 | 28 |
| 3.14E-03 | 8.56E-05 | 43.5 | 26 |
| 3.25E-03 | 4.44E-06 | 45 | 11 |
| 1.80E-03 | 3.36E-07 | 37 | 1 |
| 6.92E-03 | 3.61E-06 | 60 | 8 |
| 4.14E-03 | 3.09E-06 | 51 | 7 |
| 5.41E-03 | 6.33E-07 | 56 | 2 |
| 4.30E-03 | 1.50E-04 | 52 | 29 |
| 5.96E-03 | 7.12E-07 | 59 | 3 |
| 2.82E-03 | 4.15E-05 | 42 | 22 |

| Rotated Weierstrass ($f_{11}$) | | | | |
|---|---|---|---|---|
| # | CLPSO | PCLPSO | Sıra CLPSO | Sıra PCLPSO |
| 1 | 5.90E+00 | 4.31E+00 | 58 | 45 |
| 2 | 3.71E+00 | 8.25E-01 | 40 | 24 |
| 3 | 3.75E+00 | 3.74E-01 | 41 | 12 |
| 4 | 4.55E+00 | 2.90E-01 | 49 | 7 |
| 5 | 3.60E+00 | 3.15E-01 | 36 | 9 |
| 6 | 6.34E+00 | 4.68E-01 | 59 | 16 |
| 7 | 4.87E+00 | 5.65E-01 | 50 | 20 |
| 8 | 5.60E+00 | 9.74E-01 | 56 | 26 |
| 9 | 3.87E+00 | 4.55E-01 | 42 | 15 |
| 10 | 4.27E+00 | 5.64E-01 | 44 | 19 |
| 11 | 5.12E+00 | 6.03E-01 | 52 | 22 |
| 12 | 2.88E+00 | 7.65E-01 | 30 | 23 |
| 13 | 4.34E+00 | 1.38E-01 | 46 | 1 |
| 14 | 6.42E+00 | 2.11E-01 | 60 | 4 |
| 15 | 4.46E+00 | 2.79E-01 | 47.5 | 6 |
| 16 | 5.30E+00 | 2.68E-01 | 54 | 5 |
| 17 | 2.81E+00 | 1.20E+00 | 29 | 27 |
| 18 | 3.37E+00 | 2.08E-01 | 35 | 3 |
| 19 | 3.15E+00 | 4.44E-01 | 33 | 14 |
| 20 | 3.66E+00 | 1.78E-01 | 38 | 2 |
| 21 | 3.28E+00 | 1.25E+00 | 34 | 28 |
| 22 | 4.96E+00 | 3.36E-01 | 51 | 10 |
| 23 | 2.91E+00 | 3.84E-01 | 31 | 13 |
| 24 | 3.62E+00 | 5.92E-01 | 37 | 21 |
| 25 | 3.70E+00 | 4.94E-01 | 39 | 17 |
| 26 | 5.26E+00 | 5.28E-01 | 53 | 18 |
| 27 | 5.74E+00 | 8.73E-01 | 57 | 25 |
| 28 | 4.16E+00 | 2.93E+00 | 43 | 32 |
| 29 | 5.55E+00 | 3.12E-01 | 55 | 8 |
| 30 | 4.46E+00 | 3.46E-01 | 47.5 | 11 |

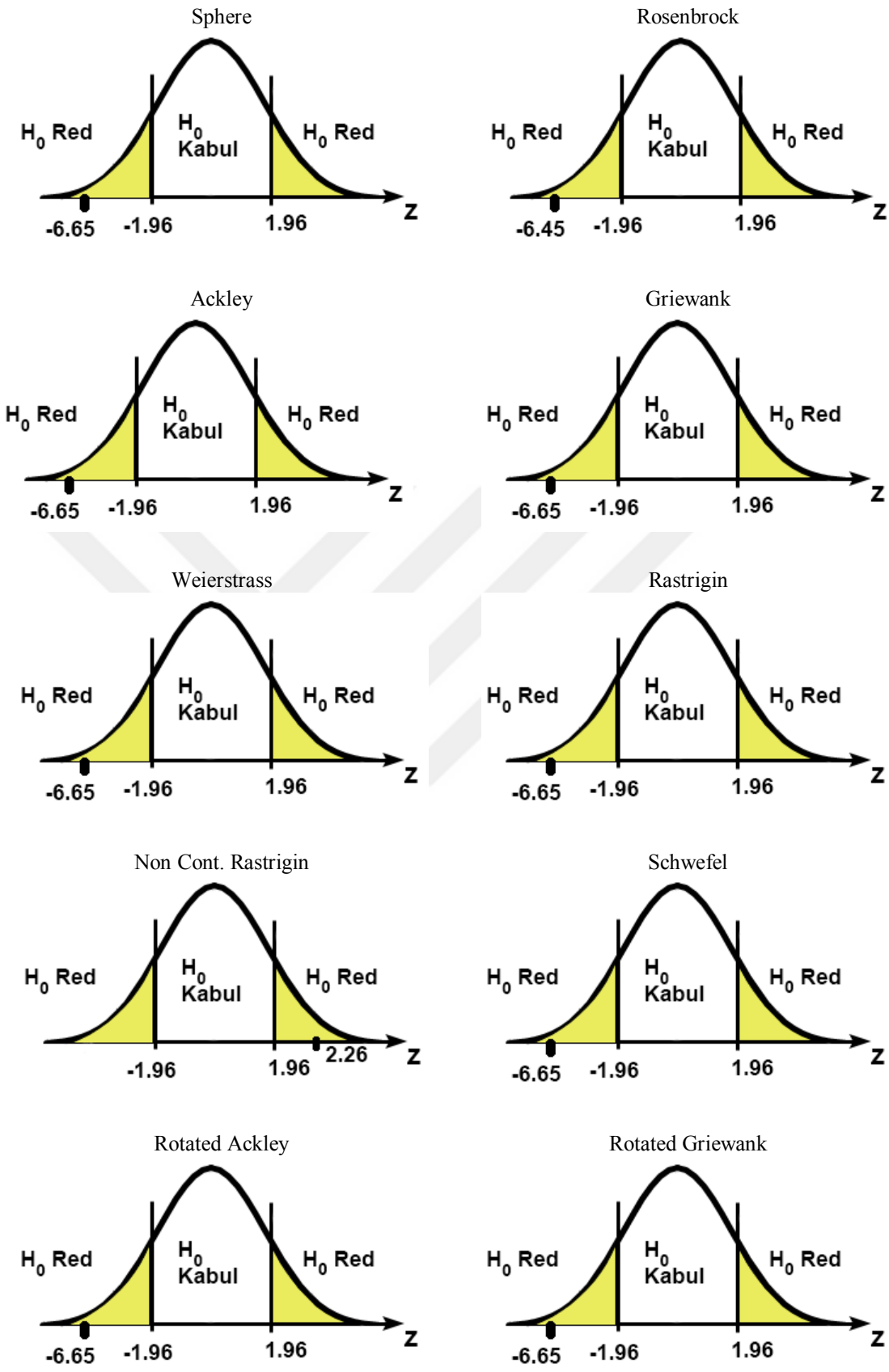
| Rotated Rastrigin ($f_{12}$) | | | |
|---|---|---|---|
| CLPSO | PCLPSO | Sıra CLPSO | Sıra PCLPSO |
| 4.50E+02 | 5.26E+02 | 5 | 56 |
| 5.06E+02 | 4.71E+02 | 43 | 16 |
| 4.91E+02 | 4.76E+02 | 29 | 20.5 |
| 5.00E+02 | 4.88E+02 | 35 | 26.5 |
| 4.52E+02 | 5.27E+02 | 6.5 | 57 |
| 5.01E+02 | 4.47E+02 | 38 | 3 |
| 4.97E+02 | 5.13E+02 | 31 | 50.5 |
| 4.98E+02 | 5.10E+02 | 32.5 | 48.5 |
| 5.06E+02 | 5.33E+02 | 43 | 59 |
| 4.76E+02 | 4.73E+02 | 20.5 | 17 |
| 4.53E+02 | 5.10E+02 | 8 | 48.5 |
| 5.09E+02 | 4.59E+02 | 47 | 9 |
| 5.37E+02 | 5.01E+02 | 60 | 38 |
| 5.04E+02 | 4.62E+02 | 40.5 | 12 |
| 4.27E+02 | 4.69E+02 | 2 | 15 |
| 5.00E+02 | 4.68E+02 | 35 | 13.5 |
| 5.06E+02 | 4.87E+02 | 43 | 24.5 |
| 5.22E+02 | 4.90E+02 | 54 | 28 |
| 5.15E+02 | 4.98E+02 | 52 | 32.5 |
| 5.01E+02 | 5.00E+02 | 38 | 35 |
| 5.08E+02 | 4.22E+02 | 45.5 | 1 |
| 5.13E+02 | 4.88E+02 | 50.5 | 26.5 |
| 4.76E+02 | 4.60E+02 | 20.5 | 10.5 |
| 4.76E+02 | 4.86E+02 | 20.5 | 23 |
| 5.29E+02 | 4.87E+02 | 58 | 24.5 |
| 4.92E+02 | 5.24E+02 | 30 | 55 |
| 4.52E+02 | 4.49E+02 | 6.5 | 4 |
| 4.74E+02 | 4.60E+02 | 18 | 10.5 |
| 5.17E+02 | 5.08E+02 | 53 | 45.5 |
| 5.04E+02 | 4.68E+02 | 40.5 | 13.5 |

| Rotated Non Cont. Rastrigin ($f_{13}$) | | | | |
|---|---|---|---|---|
| # | CLPSO | PCLPSO | Sıra CLPSO | Sıra PCLPSO |
| 1 | 3.94E+02 | 4.34E+02 | 20.5 | 41 |
| 2 | 3.54E+02 | 4.05E+02 | 6.5 | 30.5 |
| 3 | 4.03E+02 | 4.96E+02 | 28 | 60 |
| 4 | 3.57E+02 | 4.07E+02 | 8 | 33 |
| 5 | 3.74E+02 | 4.52E+02 | 12 | 48 |
| 6 | 3.61E+02 | 4.31E+02 | 9 | 40 |
| 7 | 3.97E+02 | 4.62E+02 | 22.5 | 52 |
| 8 | 3.47E+02 | 4.25E+02 | 5 | 36 |
| 9 | 3.97E+02 | 4.46E+02 | 22.5 | 46 |
| 10 | 3.15E+02 | 4.55E+02 | 1 | 49 |
| 11 | 3.91E+02 | 4.66E+02 | 19 | 55 |
| 12 | 3.82E+02 | 4.36E+02 | 17 | 42 |
| 13 | 3.80E+02 | 4.02E+02 | 13.5 | 26.5 |
| 14 | 3.94E+02 | 4.28E+02 | 20.5 | 38 |
| 15 | 3.54E+02 | 4.71E+02 | 6.5 | 56 |
| 16 | 3.62E+02 | 4.02E+02 | 10 | 26.5 |
| 17 | 4.05E+02 | 4.06E+02 | 30.5 | 32 |
| 18 | 3.81E+02 | 4.37E+02 | 15.5 | 43.5 |
| 19 | 4.04E+02 | 4.49E+02 | 29 | 47 |
| 20 | 3.81E+02 | 4.00E+02 | 15.5 | 25 |
| 21 | 3.45E+02 | 4.76E+02 | 4 | 57 |
| 22 | 3.31E+02 | 4.84E+02 | 2 | 58 |
| 23 | 3.90E+02 | 4.63E+02 | 18 | 53.5 |
| 24 | 4.18E+02 | 4.57E+02 | 35 | 50.5 |
| 25 | 3.80E+02 | 4.57E+02 | 13.5 | 50.5 |
| 26 | 3.43E+02 | 4.37E+02 | 3 | 43.5 |
| 27 | 3.64E+02 | 4.90E+02 | 11 | 59 |
| 28 | 4.17E+02 | 4.29E+02 | 34 | 39 |
| 29 | 3.98E+02 | 4.26E+02 | 24 | 37 |
| 30 | 4.63E+02 | 4.41E+02 | 53.5 | 45 |

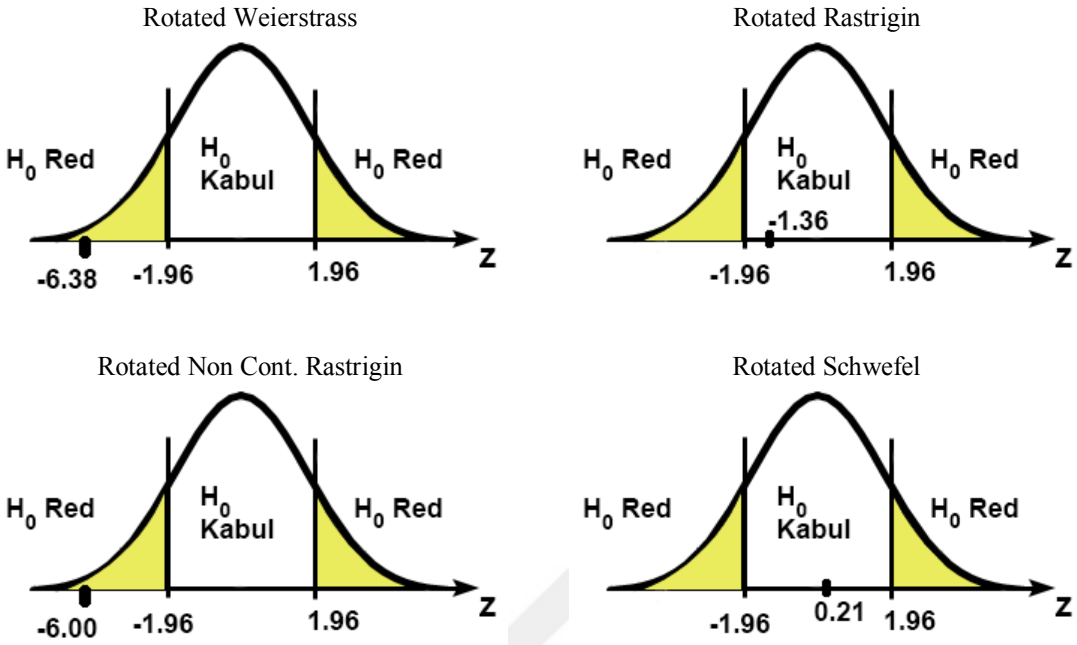
| Rotated Schwefel ($f_{14}$) | | | |
|---|---|---|---|
| CLPSO | PCLPSO | Sıra CLPSO | Sıra PCLPSO |
| 1.91E+04 | 2.07E+04 | 47.5 | 59.5 |
| 1.97E+04 | 2.07E+04 | 57.5 | 59.5 |
| 1.77E+04 | 1.92E+04 | 15 | 50 |
| 1.74E+04 | 1.75E+04 | 11 | 13 |
| 1.75E+04 | 1.80E+04 | 13 | 19 |
| 1.90E+04 | 1.97E+04 | 45.5 | 57.5 |
| 1.79E+04 | 1.90E+04 | 17 | 45.5 |
| 1.94E+04 | 1.64E+04 | 53.5 | 1 |
| 1.87E+04 | 1.88E+04 | 38.5 | 41 |
| 1.86E+04 | 1.70E+04 | 35 | 5.5 |
| 1.86E+04 | 1.68E+04 | 35 | 2.5 |
| 1.87E+04 | 1.70E+04 | 38.5 | 5.5 |
| 1.89E+04 | 1.88E+04 | 43.5 | 41 |
| 1.83E+04 | 1.84E+04 | 24.5 | 28 |
| 1.79E+04 | 1.73E+04 | 17 | 9.5 |
| 1.85E+04 | 1.95E+04 | 31.5 | 55 |
| 1.84E+04 | 1.81E+04 | 28 | 20.5 |
| 1.83E+04 | 1.86E+04 | 24.5 | 35 |
| 1.94E+04 | 1.84E+04 | 53.5 | 28 |
| 1.72E+04 | 1.73E+04 | 7.5 | 9.5 |
| 1.88E+04 | 1.86E+04 | 41 | 35 |
| 1.75E+04 | 1.82E+04 | 13 | 22.5 |
| 1.79E+04 | 1.81E+04 | 17 | 20.5 |
| 1.69E+04 | 1.92E+04 | 4 | 50 |
| 1.84E+04 | 1.82E+04 | 28 | 22.5 |
| 1.72E+04 | 1.86E+04 | 7.5 | 35 |
| 1.92E+04 | 1.96E+04 | 50 | 56 |
| 1.84E+04 | 1.68E+04 | 28 | 2.5 |
| 1.85E+04 | 1.91E+04 | 31.5 | 47.5 |
| 1.89E+04 | 1.93E+04 | 43.5 | 52 |

**Çizelge EK-1.5.** $f_1$ - $f_{14}$ için Wilcoxon sıra toplam testi istatistiği

| $f$ | $Z$ | Alt kritik değer | Üst kritik değer | $H_0$ hipotezi |
|---|---|---|---|---|
| $f_1$ | -6.65 | -1.96 | 1.96 | Ret |
| $f_2$ | -6.45 | -1.96 | 1.96 | Ret |
| $f_3$ | -6.65 | -1.96 | 1.96 | Ret |
| $f_4$ | -6.65 | -1.96 | 1.96 | Ret |
| $f_5$ | -6.65 | -1.96 | 1.96 | Ret |
| $f_6$ | -6.65 | -1.96 | 1.96 | Ret |
| $f_7$ | 2.26 | -1.96 | 1.96 | Ret |
| $f_8$ | -6.65 | -1.96 | 1.96 | Ret |
| $f_9$ | -6.65 | -1.96 | 1.96 | Ret |
| $f_{10}$ | -6.65 | -1.96 | 1.96 | Ret |
| $f_{11}$ | -6.38 | -1.96 | 1.96 | Ret |
| $f_{12}$ | -1.36 | -1.96 | 1.96 | Kabul |
| $f_{13}$ | -6.00 | -1.96 | 1.96 | Ret |
| $f_{14}$ | 0.21 | -1.96 | 1.96 | Kabul |




**Şekil EK-1.1.** $f_1$ - $f_{10}$ için Wilcoxon sıra toplam testi karar modelleri



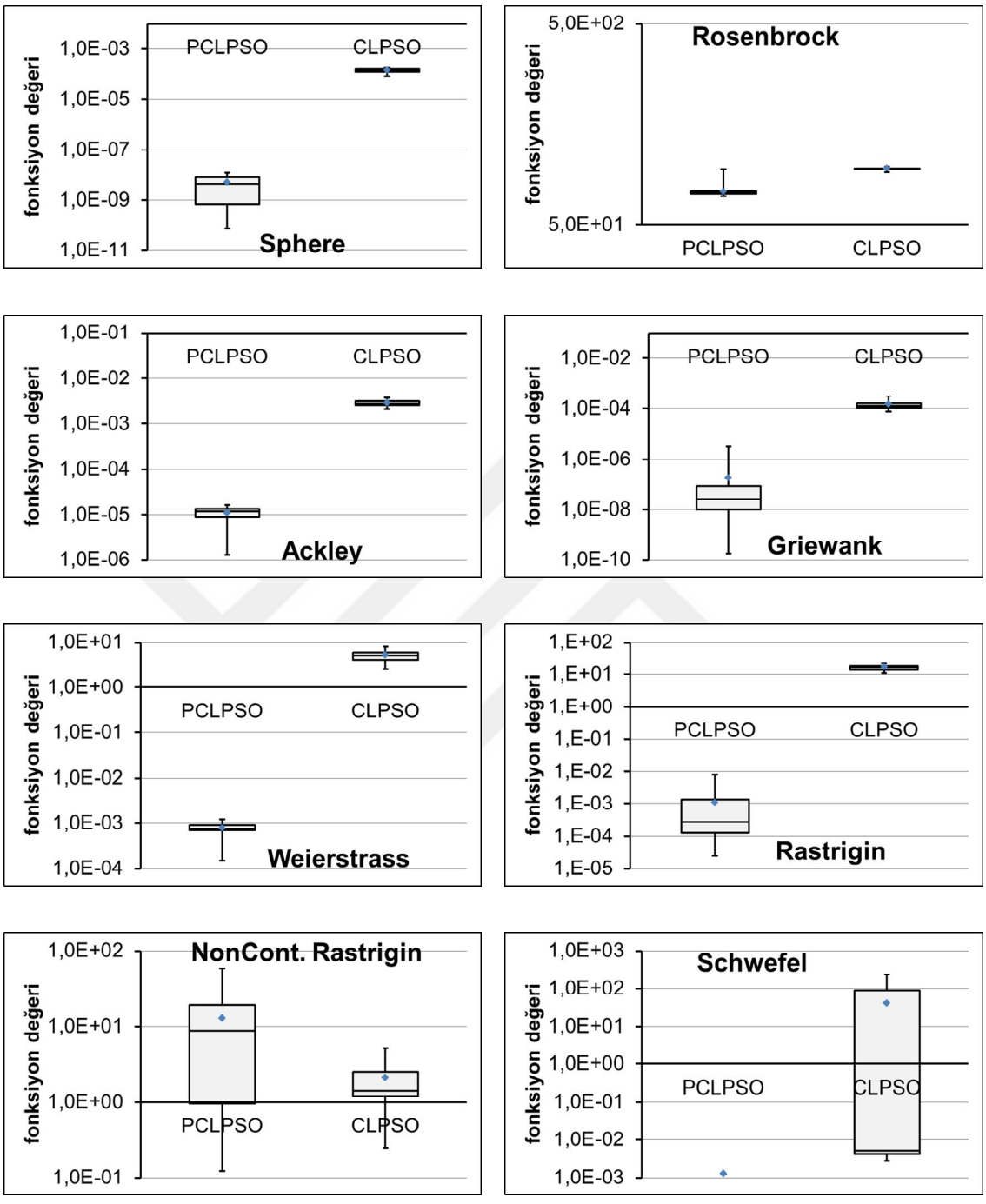

**Şekil EK-1.2.** $f_{11}$ - $f_{14}$ için Wilcoxon sıra toplam testi karar modelleri

**EK-2** Kruskal-Wallis testine ait çizelge

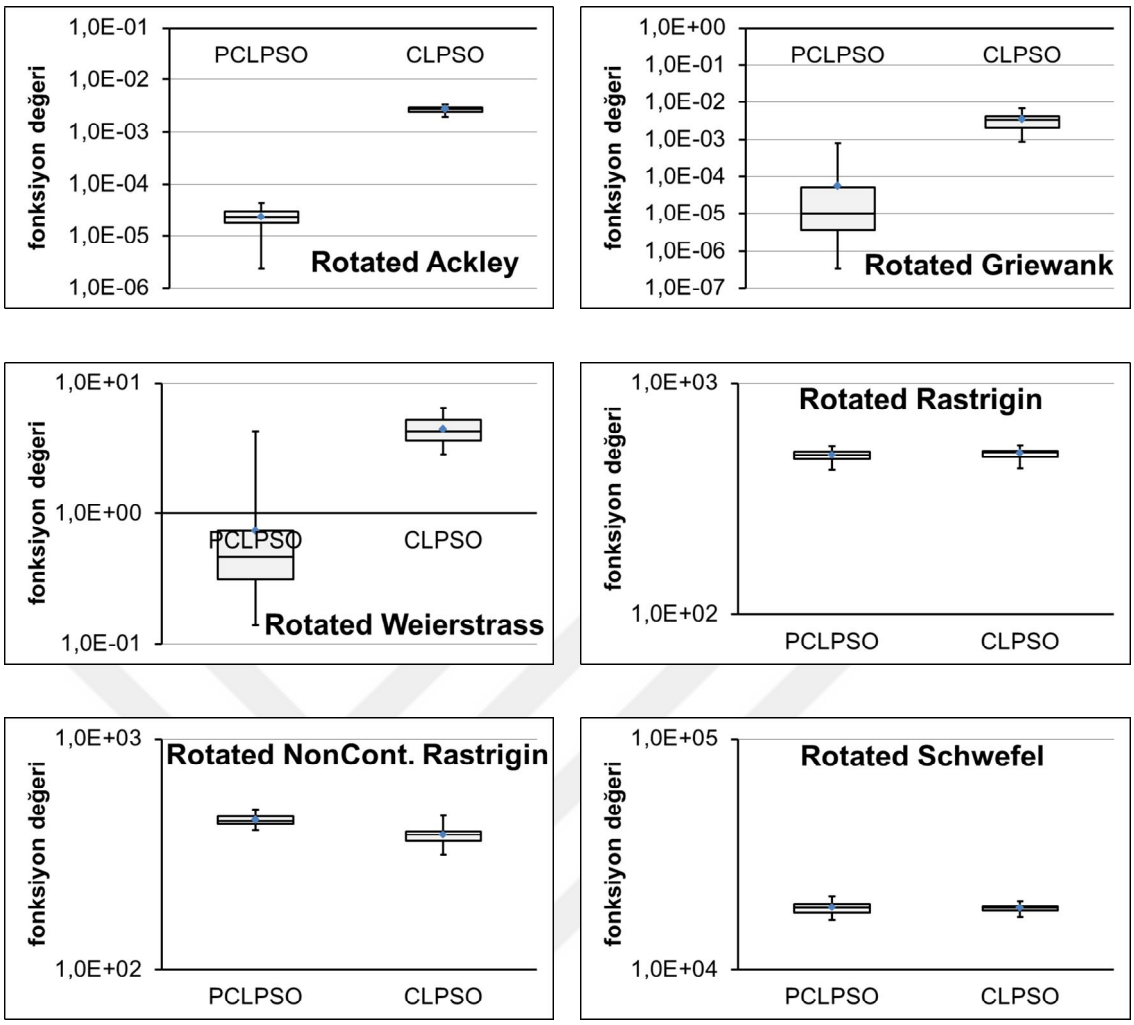
**Çizelge EK-2.1.** Kritik $\chi^2$ değerleri (Kartal, 2014)

| s.d | Önem seviyesi ($\alpha$) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0.995 | 0.990 | 0.975 | 0.950 | 0.900 | 0.100 | 0.050 | 0.025 | 0.010 | 0.005 |
| 1 | 0.000 | 0.000 | 0.001 | 0.004 | 0.016 | 2.71 | 3.84 | 5.02 | 6.63 | 7.88 |
| 2 | 0.010 | 0.020 | 0.051 | 0.103 | 0.211 | 4.61 | 5.99 | 7.38 | 9.21 | 10.60 |
| 3 | 0.072 | 0.115 | 0.216 | 0.352 | 0.584 | 6.25 | 7.81 | 9.35 | 11.34 | 12.84 |
| 4 | 0.207 | 0.297 | 0.484 | 0.711 | 1.064 | 7.78 | 9.49 | 11.14 | 13.28 | 14.86 |
| 5 | 0.412 | 0.554 | 0.831 | 1.145 | 1.61 | 9.24 | 11.07 | 12.83 | 15.09 | 16.75 |
| 6 | 0.676 | 0.872 | 1.24 | 1.64 | 2.20 | 10.64 | 12.59 | 14.45 | 16.81 | 18.55 |
| 7 | 0.989 | 1.24 | 1.69 | 2.17 | 2.83 | 12.02 | 14.07 | 16.01 | 18.48 | 20.28 |
| 8 | 1.34 | 1.65 | 2.18 | 2.73 | 3.49 | 13.36 | 15.51 | 17.53 | 20.09 | 21.96 |
| 9 | 1.73 | 2.09 | 2.70 | 3.33 | 4.17 | 14.68 | 16.92 | 19.02 | 21.67 | 23.56 |
| 10 | 2.16 | 2.56 | 3.25 | 3.94 | 4.87 | 15.99 | 18.31 | 20.48 | 23.21 | 25.19 |
| 11 | 2.60 | 3.05 | 3.82 | 4.57 | 5.58 | 17.28 | 19.68 | 21.92 | 24.72 | 26.76 |
| 12 | 3.07 | 3.57 | 4.40 | 5.23 | 6.30 | 18.55 | 21.03 | 23.34 | 26.22 | 28.30 |
| 13 | 3.57 | 4.11 | 5.01 | 5.89 | 7.04 | 19.81 | 22.36 | 24.74 | 27.69 | 29.82 |
| 14 | 4.07 | 4.66 | 5.63 | 6.57 | 7.79 | 21.06 | 23.68 | 26.12 | 29.14 | 31.32 |
| 15 | 4.60 | 5.23 | 6.26 | 7.26 | 8.55 | 22.31 | 25.00 | 27.49 | 30.58 | 32.80 |
| 16 | 5.14 | 5.81 | 6.91 | 7.96 | 9.31 | 23.54 | 26.30 | 28.85 | 32.00 | 34.27 |
| 17 | 5.70 | 6.41 | 7.56 | 8.67 | 10.09 | 24.77 | 27.59 | 30.19 | 33.41 | 35.72 |
| 18 | 6.26 | 7.01 | 8.23 | 9.39 | 10.86 | 25.99 | 28.87 | 31.53 | 34.81 | 37.16 |
| 19 | 6.84 | 7.63 | 8.91 | 10.12 | 11.65 | 27.20 | 30.14 | 32.85 | 36.19 | 38.58 |
| 20 | 7.43 | 8.26 | 9.59 | 10.85 | 12.44 | 28.41 | 31.41 | 34.17 | 37.57 | 40.00 |
| 21 | 8.03 | 8.90 | 10.28 | 11.59 | 13.24 | 29.62 | 32.67 | 35.48 | 38.93 | 41.40 |
| 22 | 8.64 | 9.54 | 10.98 | 12.34 | 14.04 | 30.81 | 33.92 | 36.78 | 40.29 | 42.80 |
| 23 | 9.26 | 10.20 | 11.69 | 13.09 | 14.85 | 32.01 | 35.17 | 38.08 | 41.64 | 44.18 |
| 24 | 9.89 | 10.86 | 12.40 | 13.85 | 15.66 | 33.20 | 36.42 | 39.36 | 42.98 | 45.56 |
| 25 | 10.52 | 11.52 | 13.12 | 14.61 | 16.47 | 34.38 | 37.65 | 40.65 | 44.31 | 46.93 |
| 26 | 11.16 | 12.20 | 13.84 | 15.38 | 17.29 | 35.56 | 38.89 | 41.92 | 45.64 | 48.29 |
| 27 | 11.81 | 12.88 | 14.57 | 16.15 | 18.11 | 36.74 | 40.11 | 43.19 | 46.96 | 49.64 |
| 28 | 12.46 | 13.56 | 15.31 | 16.93 | 18.94 | 37.92 | 41.34 | 44.46 | 48.28 | 50.99 |
| 29 | 13.12 | 14.26 | 16.05 | 17.71 | 19.77 | 39.09 | 42.56 | 45.72 | 49.59 | 52.34 |
| 30 | 13.79 | 14.95 | 16.79 | 18.49 | 20.60 | 40.26 | 43.77 | 46.98 | 50.89 | 53.67 |
| 34 | 20.71 | 22.16 | 24.43 | 26.51 | 29.05 | 51.80 | 55.76 | 59.34 | 63.69 | 66.77 |
| 50 | 27.99 | 29.71 | 32.36 | 34.76 | 37.69 | 63.17 | 67.50 | 71.42 | 76.15 | 79.49 |
| 60 | 35.53 | 37.48 | 40.48 | 43.19 | 46.46 | 74.40 | 79.08 | 83.30 | 88.38 | 91.95 |
| 70 | 48.28 | 45.44 | 48.76 | 51.74 | 55.33 | 85.53 | 90.53 | 95.02 | 100.4 | 104.22 |
| 80 | 51.17 | 53.54 | 57.15 | 60.39 | 64.28 | 96.58 | 101.9 | 106.6 | 112.3 | 116.32 |
| 90 | 59.20 | 61.75 | 65.65 | 69.13 | 73.29 | 107.6 | 113.1 | 118.1 | 124.1 | 128.3 |
| 100 | 67.33 | 70.06 | 74.22 | 77.93 | 82.36 | 118.5 | 124.3 | 129.6 | 135.8 | 140.2 |

**EK-3** Kıyaslama fonksiyonlarının kutu grafiği

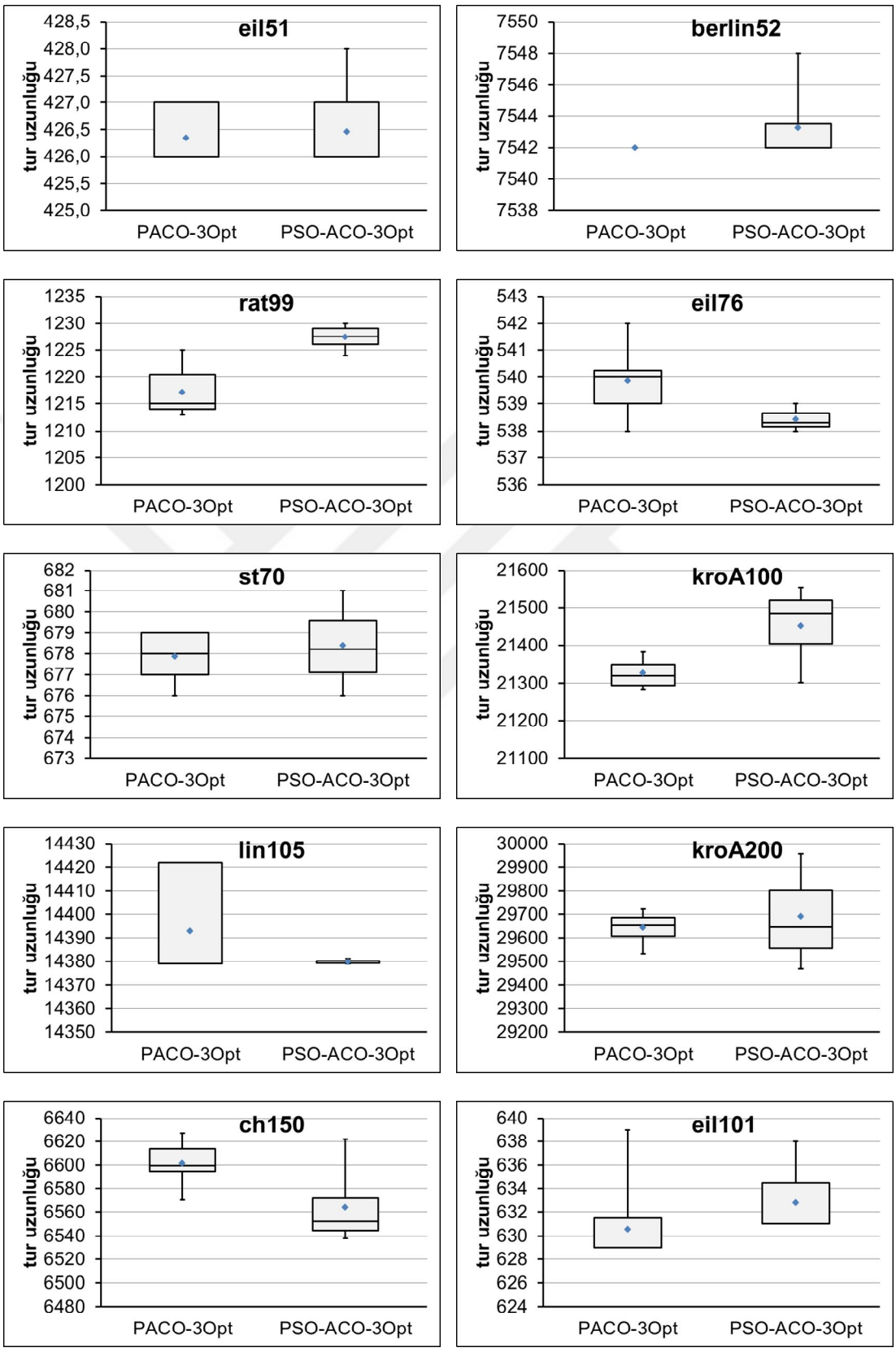



**Şekil EK-3.1.** 100-B Kıyaslama fonksiyonlarının kutu grafiği

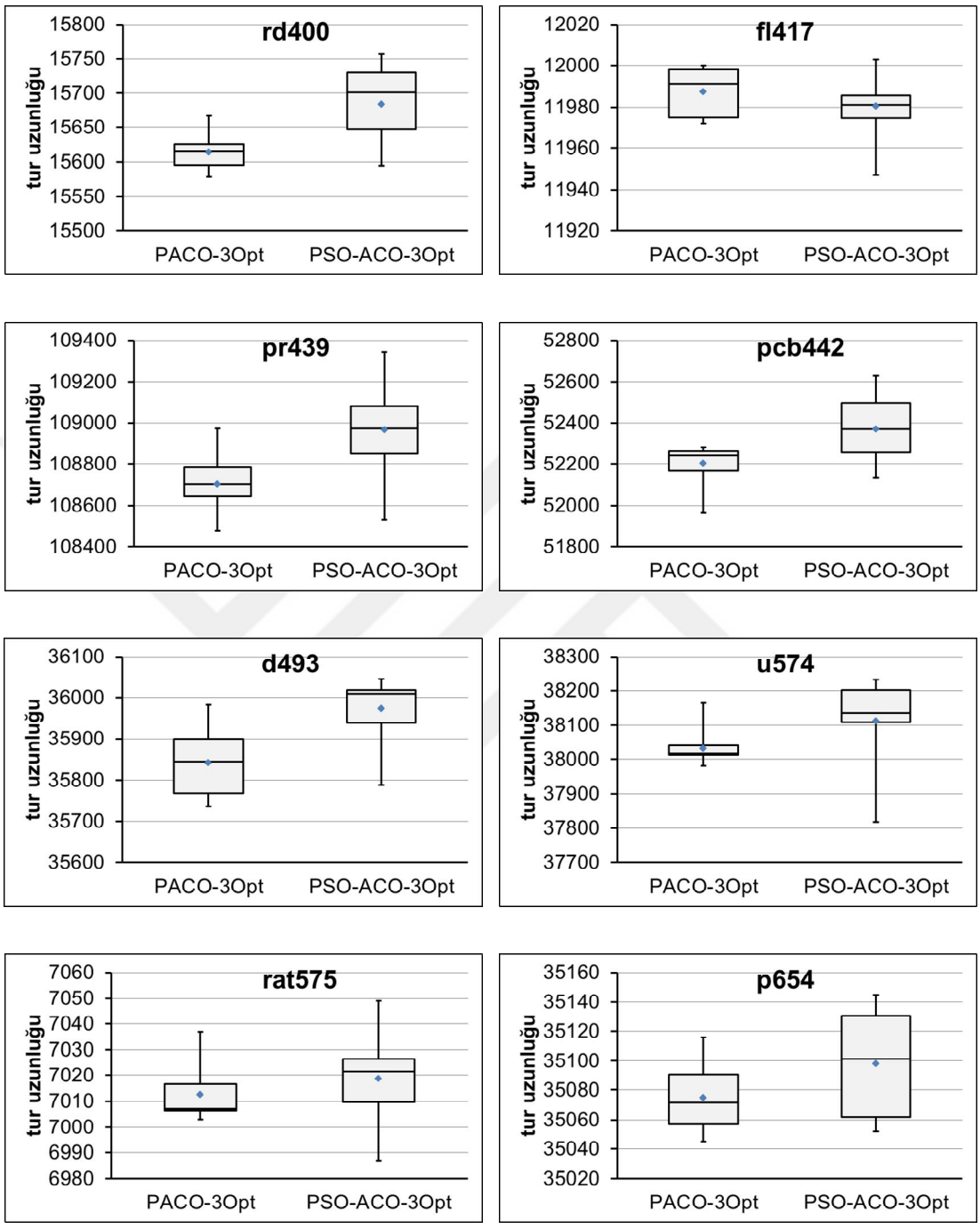



**Şekil EK-3.1.** 100-B Kıyaslama fonksiyonlarının kutu grafiği (devamı)

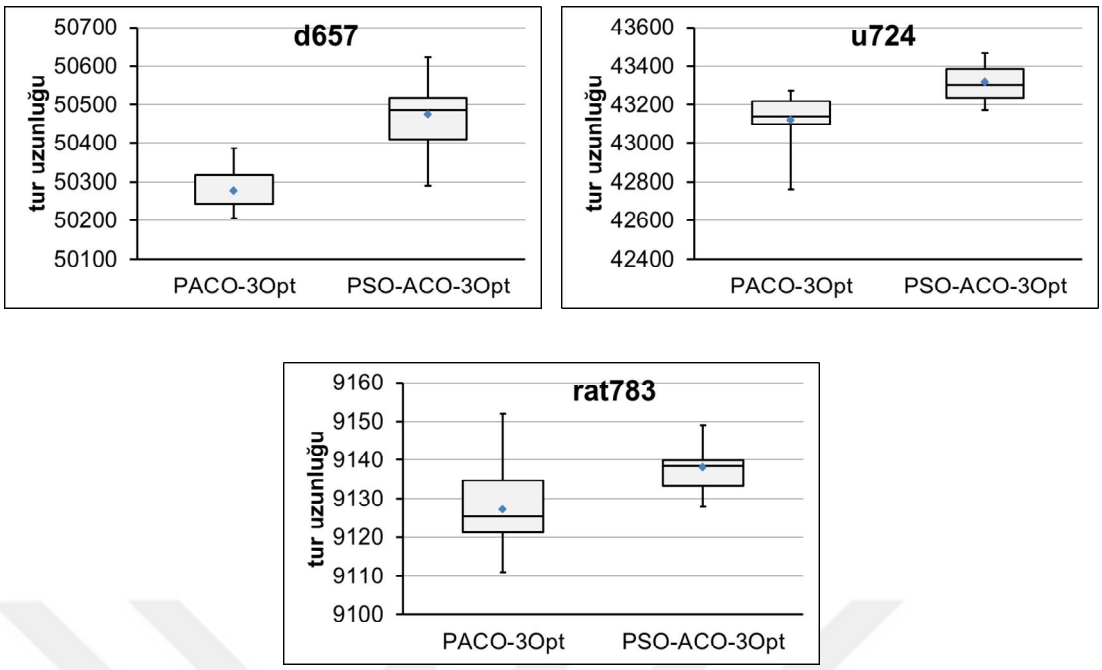
**EK-4** GSP örneklerinin kutu grafiği




**Şekil EK-4.1.** Küçük-ölçekli GSP örneklerinin kutu grafiği




**Şekil EK-4.2.** Büyük-ölçekli GSP örneklerinin kutu grafiği




**Şekil EK-4.2.** Büyük-ölçekli GSP örneklerinin kutu grafiği (devamı)

# ÖZGEÇMİŞ

## KİŞİSEL BİLGİLER


| | | |
|---|---|---|
| **Adı Soyadı** | **:** | Şaban GÜLCÜ |
| **Uyruğu** | **:** | T.C. |
| **Doğum Yeri ve Tarihi** | **:** | Alanözü/Karaman, 01.10.1982 |
| **Telefon** | **:** | +90 554 323 23 56 |
| **Faks** | **:** | |
| **e-mail** | **:** | sgulcu@konya.edu.tr |


## EĞİTİM

| **Derece** | | **Adı, İlçe, İl** | **Bitirme Yılı** |
|---|---|---|---|
| Lise | : | Konya İmam-Hatip Lisesi | 1999 |
| Üniversite | : | Viyana Teknik Üniversitesi, Bilgisayar Bilimleri Fakültesi, Yazılım Mühendisliği Bölümü | 2006 |
| Yüksek Lisans | : | Viyana Teknik Üniversitesi, Bilgisayar Bilimleri Fakültesi, Yazılım Mühendisliği Bölümü | 2007 |
| Doktora | : | Selçuk Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği A.B.D. | |

## İŞ DENEYİMLERİ

| **Yıl** | **Kurum** | **Görevi** |
|---|---|---|
| 2006-2007 | LKSM GmbH | Yazılım Mühendisi |
| 2008-2010 | Belbim A.Ş. | Yazılım Mühendisi |
| 2010-2011 | KTO Karatay Üniversitesi | Araştırma Görevlisi |
| 2011-devam | Necmettin Erbakan Üniversitesi | Öğretim Görevlisi |

**UZMANLIK ALANI:** Metasezgisel algoritmalar, optimizasyon, paralel hesaplama, dağıtık sistemler, yazılım mühendisliği

**YABANCI DİLLER:** İngilizce, Almanca

## YAYINLAR

### SCI, SSCI, AHCI indekslerine giren dergilerde yayınlanan makaleler


Gülcü Ş, Mahi M, Baykan Ö K, Kodaz H, 2016. A parallel cooperative hybrid method based on Ant Colony Optimization and 3-Opt algorithm for solving Traveling Salesman Problem. Soft Computing, doi:10.1007/s00500-016-2432-3. (Doktora Tezinden)

Gülcü Ş, Kodaz H, 2015. A novel parallel multi-swarm algorithm based on comprehensive learning particle swarm optimization. Engineering Applications of Artificial Intelligence, vol. 45, pp. 33-45. (Doktora Tezinden)

**Diğer dergilerde yayınlanan makaleler**

Gülcü Ş, Kodaz H, 2016. Analysing the Migration Period Parameter in Parallel Multi-Swarm Particle Swarm Optimization, International Journal of Computer Science and Information Technology, vol. 8, issue 3, pp. 31-45. (Doktora Tezinden)

Dayıoğlu Gülcü S, Gülcü Ş, Kahramanlı H, 2014. Solution of Travelling Salesman Problem using Ant Colony Algorithm, Applied Mechanics and Materials, vol. 548-549, pp. 1206-1212.

**Hakemli konferans/sempozyumların bildiri kitaplarında yer alan yayınlar**

Gülcü Ş, Kodaz H, Effects of the different migration periods on parallel multi-swarm PSO, 6th International Conference on Computer Science, Engineering and Information Technology (CCSEIT 2016), pp. 13-22, 21-22 May 2016, Vienna, Austria. (Doktora Tezinden)

Dayıoğlu Gülcü S, Gülcü Ş, Kahramanlı H, Solution of Travelling Salesman Problem using Intelligent Water Drops Algorithm, 2nd International Conference on Information Technology and Computer Networks (ITCN'13), 08-10 October 2013, Antalya, Turkey.

Gülcü Ş, Dayıoğlu Gülcü S, Improving of the LSB Based Steganography Technique using Multiple LSBs and Discrete Logarithm, 1st International Symposium on Digital Forensics and Security (ISDFS'13), 20-21 May 2013, Elazığ, Turkey.

Gülcü Ş, Ülker E, Knot Estimation of the B-Spline Curve with Strength Pareto Evolutionary Algorithm 2 (SPEA2), Proc. of the 16th Int. Conf. on Computers (part of CSCC'12), pp. 308-314, Kos Island, Greece, 14-17 July, 2012.