T.C.
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BULUT SİSTEMLERDE GÖREV ÇİZELGELEME PROBLEMLERİNE METASEZGİSEL BİR ÇÖZÜM MODELİNİN GELİŞTİRİLMESİ

Mücahit BÜRKÜK

Yüksek Lisans Tezi

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

Donanım Bilim Dalı

TEMMUZ 2022

T.C.
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

Bilgisayar Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

BULUT SİSTEMLERDE GÖREV ÇİZELGELEME PROBLEMLERİNE METASEZGİSEL BİR ÇÖZÜM MODELİNİN GELİŞTİRİLMESİ

Tez Yazarı
Mücahit BÜRKÜK

Danışman
Dr. Öğr. Üyesi Güngör YILDIRIM

TEMMUZ 2022
ELAZIĞ

T.C.
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

Bilgisayar Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

Başlığı:	Bulut Sistemlerde Görev Çizelgeleme Problemlerine Metasezgisel Bir Çözüm Modelinin Geliştirilmesi
Yazarı:	Mücahit BÜRKÜK
İlk Teslim Tarihi:	03.06.2022
Savunma Tarihi:	01.07.2022

TEZ ONAYI

Fırat Üniversitesi Fen Bilimleri Enstitüsü tez yazım kurallarına göre hazırlanan bu tez aşağıda imzaları bulunan jüri üyeleri tarafından değerlendirilmiş ve akademik dinleyicilere açık yapılan savunma sonucunda OYBİRLİĞİ ile kabul edilmiştir.

		İmza	
Danışman:	Dr. Öğr. Üyesi Güngör YILDIRIM Fırat Üniversitesi, Mühendislik Fakültesi		Onayladım
Başkan:	Doç. Dr. İlhan AYDIN Fırat Üniversitesi, Mühendislik Fakültesi		Onayladım
Üye:	Dr. Öğr. Üyesi Alper KILIÇ Konya Teknik Üniversitesi, Mühendislik ve Doğa Bilimleri Fakültesi		Onayladım

Bu tez, Enstitü Yönetim Kurulunun ...../...../20..... tarihli toplantısında tescillenmiştir.

İmza

Prof. Dr. Kürşat Esat ALYAMAÇ
Enstitü Müdürü

BEYAN

Fırat Üniversitesi Fen Bilimleri Enstitüsü tez yazım kurallarına uygun olarak hazırladığım "Bulut Sistemlerde Görev Çizelgeleme Problemlerine Metasezgisel Bir Çözüm Modelinin Geliştirilmesi" Başlıklı Yüksek Lisans Tezimin içindeki bütün bilgilerin doğru olduğunu, bilgilerin üretilmesi ve sunulmasında bilimsel etik kurallarına uygun davrandığımı, kullandığım bütün kaynakları atıf yaparak belirttiğimi, maddi ve manevi desteği olan tüm kurum/kuruluş ve kişileri belirttiğimi, burada sunduğum veri ve bilgileri unvan almak amacıyla daha önce hiçbir şekilde kullanmadığımı beyan ederim.

01.07.2022

Mücahit BÜRKÜK

ÖNSÖZ

Bulut teknolojileri, son yıllarda araştırmacıların, uzmanların ve şirketlerin birçok sorununu kolayca çözmelerine yardımcı olmuştur. Kullandığın kadar öde, sanallaştırma ile verimli kaynak rezervasyonu, büyük veri işleme gibi birçok faydalı özellik, bulut teknolojilerini kısa sürede cazip hale getirdi. Ancak her gelen kolaylık ile beraber arka planda yürütülmesi ve çözülmesi gereken sorunları da gün yüzüne çıkardı. Bulut sistemler, çok pahalı alt yapı bileşenlerine sahip, ciddi enerji sarfiyatı olan, belirli bir hizmet kalitesi vadeden başka bir değişle optimum işletme şartlarını her zaman önceleyen yapılardır. Optimum çalışma konusunda en önemli başlıklardan biri verimli görev çizelgelemelerinin yapılabilmesidir. Başarılı bir görev çizelgeleme beraberinde hem müşteri hem de bulut servis sağlayıcı için ekonomik faydalar sağlayacaktır. Her ne kadar verimli görev çizelgeleme farklı amaçlar için yapılabilse de her başarılı görev çizelgeleme senaryosu için önemli bir avantaj garanti edilebilmektedir. Bu avantajlara müşteri açısından bakıldığında optimum kaynak kullanımı ile daha uygun maliyet, sağlayıcı tarafında ise daha az enerji tüketimi ve başarılı hizmet garantisi ilk olarak kendini gösterecektir.

Doğası gereği NP-hard olan görev çizelgeleme (dağıtımı) problemleri için farklı yaklaşımlar önerilse de metasezgisel çözümler her zaman güncelliklerini korumuştur. Rastgele arama temelli optimizasyon yaklaşımların esnek doğası, en iyi çözümü garanti edemez ancak görev çizelgeleme gibi karmaşık problemlerde diğer yöntemlerin sağlayamadığı başarımı gösterebilir. Bu tez çalışması da bu avantajı kullanarak bulut sistemlerde görev çizelgeleme için metasezgisel alternatif çözümler üretmeye odaklanmıştır.

Bu çalışmaları gerçekleştirirken, sorularımı özveri ile cevaplayan, sahip olduğu tüm akademik kaynakları tarafımla paylaşan danışman hocam Sayın Dr. Öğr. Üyesi Güngör YILDIRIM' a katkılarından dolayı çok teşekkür ederim. Çalışmalarım süresince, her koşulda yanımda olan tüm zorlukları benimle göğüsleyen sevgili Eşime, benden bir an olsun yardımını esirgemeyen kıymetli Ablama, hayatımın her evresinde asla desteklerini esirgemeyen ve bugünlere gelmemi sağlayan Babam, Annem, Abim ve Kardeşime şükranlarımı sunarım.

Mücahit BÜRKÜK
ELAZIĞ, 2022

İÇİNDEKİLER

Sayfa

ÖNSÖZ iv
İÇİNDEKİLER v
ÖZET vii
ABSTRACT viii
ŞEKİLLER LİSTESİ ix
TABLOLAR LİSTESİ xi
SİMGELER VE KISALTMALAR xii

1. GİRİŞ 1

2. TEMEL KAVRAMLAR VE KAPSAM 7

2.1. Optimizasyon 7
2.1.1. Analitik Algoritmalar 8
2.1.2. Sezgisel Algoritmalar 8
2.1.3. Metasezgisel Yöntemler 8
2.2. Bulut Bilişim Sistemleri 9
2.2.1. Bulut Bilişim Hizmet Modelleri 10
2.2.2. Yük Dengeleyici (Load Balancing) 12
2.2.3. Sanallaştırma 13
2.2.4. Görev Çizelgeleme (Task Scheduling) 14
2.2.5. Simülasyon Ortamı ve CloudSim 16

3. MATERYAL VE METOT 18

3.1. Denizanası Arama Optimizasyonu (JSO) 19
3.1.1. Optimizasyonun Esin Kaynağı 19
3.1.2. JSO' nun Matematiksel Modeli 20
3.2. Klasik JSO' nun İlk Performans Testleri ve Kullanılan Test (Benchmark) Fonksiyonları 25
3.2.1. Sphere İşlevi 26
3.2.2. Rastrigin İşlevi 26
3.2.3. Styblinski-Tang İşlevi 26
3.2.4. Rosenbrock İşlevi 27
3.3. Klonlanabilir JSO (C-JSO) ve Görev Çizelgelemeye Uygulanması 27
3.4. Görev Çizelgeleme Paralel Çalışma 29
3.4.1. Çoklu İş Parçacığı (Multi-Thread) Temelli JSO 29
3.4.2. Çoklu Süreç (Multi-Process) 32
3.4.3. Benzerlik Kontrollü Multi Process 34

4. BULGULAR VE TARTIŞMA 35

4.1. Klasik Denizanası Algoritması (JSO) ve Benchmark Testleri 35
4.2. Görev Çizelgeleme ve Gerekli Simülasyon Parametreleri 36
4.3. Klasik Denizanası Optimizasyonu ile Görev Çizelgeleme (JSO) 37
4.4. Klonlanabilir Denizanası Algoritması ile Görev Çizelgeleme (C-JSO) 38
4.5. Görev Çizelgelemede Kullanılan Tüm Yöntemlerin Karşılaştırılması 41
4.6. Görev Çizelgelemede Paralel Yaklaşım Deneyleri 42
4.6.1. Önerilen Paralel JSO Modeli 45
4.6.2. Önerilen Paralel JSO' nun Karşılaştırmalı Sonuçları 48

5. SONUÇLAR 51
KAYNAKLAR 52
ÖZGEÇMİŞ

ÖZET

Bulut Sistemlerde Görev Çizelgeleme Problemlerine Metasezgisel Bir Çözüm Modelinin Geliştirilmesi

Mücahit BÜRKÜK

Yüksek Lisans Tezi

FIRAT ÜNİVERSİTESİ
Fen Bilimleri Enstitüsü

Bilgisayar Mühendisliği Anabilim Dalı

Temmuz 2022, Sayfa: xii + 54

Bulut bilişim teknolojisi birçok verinin internet ortamında barınmasını, değerlendirilebilmesini, kullanıcıların yalnızca tükettiği hizmet kadar ödeme yapmalarını sağlayan sanallaştırılabilir ve ölçeklenebilir kaynakların toplamıdır. İnternet altyapısının gelişmesi, Nesnelerin İnterneti teknolojisinin yaygınlaşması, büyük verinin hızlı artışı ve buna yönelik çalışmaların ortaya çıkması, yapay zekâ çalışmalarındaki gelişmeler gibi birçok sebep bulut teknolojilerinin yaygınlaşmasına neden olmuştur. Bulut bilişimin en önemli mekanizmalarından biri sanal makinelerdir. Sanal makineler müşterilerin ihtiyaçları doğrultusunda bulut sistem üzerindeki kaynaklardan oluşturulurlar. İş hacmine bağlı olarak müşterilerin sanal makine sayıları ve özellikleri çeşitlilik gösterebilir. Müşteriler bu sanal makinelerin özelliklerine ve kullanım sürelerine göre bulut sağlayıcıya belirli ücretler öderler. Sanal makineler üzerinde çalıştırılması gereken görevlerin yanlış çizelgelenmesi görev tamamlanma süresinin (makespan) artmasına ve doğal olarak da müşteri için maliyet artışına yol açar. Görev tamamlanma süresindeki bu artış, dolaylı olarak bulut sağlayıcının da enerji sarfiyatı ve bakım onarım maliyetlerini olumsuz etkiler. Bu sebeple bulut sitemlerde sanal makineler için iyi bir görev çizelgeleme algoritmasının kullanılması hem müşteri hem de bulut sağlayıcı açısından zorunludur.

Görev çizelgeleme, NP-hard tipi bir problemdir. Deterministtik yaklaşımlar yerine metasezgisel algoritmaların kullanılması performans açısından bu tip problemlerin çözümünde sıklıkla tercih edilmektedir. Ancak öte yandan, problem tipinden dolayı, rastgele arama temelli metasezgisel algoritmaların lokal minimalara takılma olasılıkları da yüksektir. Bu olasılık görev ve sanal makine sayılarının artmasıyla daha da artabilmektedir. Bu sebeple kullanılan metasezgisel algoritmaların bu sorunu aşacak mekanizmalar kullanması gerekmektedir. Bu çalışma bu sorunun çözümü için farklı bir yaklaşım mekanizması kullanan ve güncel metasezgisel algoritmalardan olan Denizanası Arama Optimizasyonu (Jellyfish Search Optimizer) temelli bir çözüm önermektedir. Önerilen yöntemin en özgün yanı, daha hızlı bir şekilde lokal minimalardan kurtulmak için farklı bir benzerlik kontrolü ile dinamik popülasyon artışına imkân vermesidir. Böylece arama uzayında daha verimli bir keşif süreci gerçekleştirilmiş olmaktadır. Buna ek olarak bu algoritmanın, görev çizelgeleme problemi için çoklu iş parçacığı (Multi-Thread) ve çoklu-süreç (Multi-Process) analizleri de bu tez kapsamında yapılmıştır. Önerilen yöntemin performansı, CloudSim simülatöründe farklı senaryolar için karşılaştırmalı olarak denenmiş ve ispatlanmıştır.

Anahtar Kelimeler: Görev planlama, Bulut bilişim, Metasezgisel, Denizanası Arama Optimizasyonu

ABSTRACT

Developing a Metaheuristic Solution Model to Task Scheduling Problems in Cloud Systems

Mücahit BÜRKÜK

Master's Thesis

FIRAT UNIVERSITY
Graduate School of Natural and Applied Sciences

Department of Computer Engineering

July 2022, Pages: xii + 54

Cloud computing technology is the sum of virtualizable and scalable resources that enable data to be hosted and evaluated on the internet, and users to pay only for the resources they consume. Many reasons, such as the development of the Internet infrastructure, the spread of the Internet of Things technology, the rapid growth of big data and the emergence of studies on this, and the developments in artificial intelligence studies, have led to the widespread use of cloud technologies. One of the most important mechanisms of cloud computing is virtual machines. Virtual machines are created from resources on the cloud system in line with the needs of customers. Depending on the business volume, the number of virtual machines created by customers and their features may vary. Customers pay certain fees to the cloud provider based on the features and duration of use of these virtual machines. Incorrect scheduling of tasks that need to be run on virtual machines leads to increased task completion time (makespan), and naturally, increased cost for the customer. This increase in task completion time indirectly affects the energy consumption and maintenance costs of the cloud provider. For this reason, the use of a good task scheduling algorithm in cloud systems is mandatory for both the customer and the cloud provider.

Task scheduling is an NP-hard type problem. The use of metaheuristic algorithms instead of deterministic approaches is often preferred in solving such problems in terms of performance. However, due to the type of problem, metaheuristic algorithms based on random search may get stuck in local minimums. This possibility may increase in case where the number of tasks and virtual machines has increased. For this reason, the metaheuristic algorithms preferred should use efficient mechanisms to overcome this problem. This study proposes a solution based on Jellyfish Search Optimizer, one of the current metaheuristic algorithms, which uses a different approach mechanism to solve this problem. The most unique aspect of the proposed method is that it allows dynamic population growth with a different similarity control to get rid of local minimums more quickly. Thus, a more efficient exploration process is realized in the search space. In addition, the multi-thread and multi-process analyses of this algorithm for the task scheduling problem were also made within the scope of this thesis. The performance of the proposed method has been comparatively tested and proven for different scenarios in the CloudSim simulator.

Keywords: Task scheduling, Cloud computing, Metaheuristic, Jellyfish Search Optimization

ŞEKİLLER LİSTESİ

Sayfa

Şekil 2.1. Optimizasyon Teknikleri 7

Şekil 2.2. Metasezgisel Algoritmaların Sınıflandırılması 9

Şekil 2 3. Bulut Bilişim Sistemlerinin Ayrıcalıkları 10

Şekil 2.4. Bulut Bilişim Sağlayıcıları Hizmet Modelleri 11

Şekil 2.5. Yük dengeleme çeşitleri 12

Şekil 2.6. Bulut sistemlerde yük dengeleme 13

Şekil 2.7. Fiziksel sunucu üzerindeki sanal makinenelerin temsili gösterimi 13

Şekil 2.8. Görev Çizelgeleme Stratejileri 15

Şekil 2.9. Görev Çizelgeleme Algoritmaları 16

Şekil 2.10. CloudSim çalışma prosedürü [11] 17

Şekil 3.1. Makespan Tanımı 18

Şekil 3.2. Denizanasının Okyanustaki Davranışları 20

Şekil 3.3. Denizanası Algoritmasının Akış Şeması [3] 23

Şekil 3.4. Zaman Kontrol Mekanizmasının Simülasyonu 24

Şekil 3.5. Yeniden Girme Yöntemi 25

Şekil 3.6. Görev Tahsisi İçin Örnek Bir Benzerlik Kontrolü 28

Şekil 3.7. Python Multi Thread çalışma prensibi 30

Şekil 3.8. Çoklu iş parçacığı (Multi-Thread) JSO çalışma prensibi 31

Şekil 3.9. Çoklu süreç (Multi-Process) JSO çalışma prensibi 33

Şekil 3.10. Multi-Threading ve Multi-Processing Ayrıcalıkları 33

Şekil 3.11. Önerilen çoklu süreç (Multi-Process) JSO çalışma prensibi 34

Şekil 4.1. Klasik JSO farklı popülasyon boyutlarına göre makespan ve süre sonuçları 37

Şekil 4.2. C-JSO makespan ve süre sonuçları 38

Şekil 4.3. Tüm görevler için artış ve benzerlik oranlarının süreye etkisi 39

Şekil 4.4. Tüm görevler için artış oranlarının makespan etkisi 39

Şekil 4.5. Tüm görevler için benzerlik oranlarının makespan etkisi 40

Şekil 4.6. C-JSO örnek simülasyon çıktıları 40

Şekil 4.7. Kullanılan yöntemlerin makespan ve süre karşılaştırması 41

Şekil 4.8. Multi-Thread ve Multi-Process makespan karşılaştırması 43

Şekil 4.9. Multi Thread ve Multi Process süre karşılaştırması 44

Şekil 4.10. Multi Process farklı popülasyon senaryoları için süre grafiği 44

Şekil 4.11. Tüm senaryolar için benzerlik oranları makespan karşılaştırması 46
Şekil 4.12. Tüm senaryolar için benzerlik oranları süre karşılaştırması 47
Şekil 4.13. Klasik Multi-Process ve benzerlik kontrollü Multi-Process yürütüm süresi grafiği 48
Şekil 4.14. Klasik Multi-Process ve benzerlik kontrollü Multi-Process makespan karşılaştırma grafiği .. 49

TABLOLAR LİSTESİ

Sayfa

Tablo 3.1. Zaman Kontrol Mekanizmasının Sözde Kodu 24

Tablo 3.2. C-JSO tabanlı görev çizelgeleme sözde kodu 29

Tablo 3.3. Multi-thread JSO tabanlı görev çizelgeleme sözde kodu 31

Tablo 3.4. Multi-process JSO tabanlı görev çizelgeleme sözde kodu 32

Tablo 4.1. Benchmark fonksiyonları deney karşılaştırması 35

Tablo 4.2. Benchmark fonksiyonları farklı senaryolar için istatiksel sonuçlar 36

Tablo 4.3. Simülasyon Parametreleri 36

Tablo 4.4. Kullanılan metotların istatistiksel sonuçları (Makespan) 42

Tablo 4.5. Klasik Multi-Process için istatistiksel sonuçlar 45

Tablo 4.6. Benzerlik Kontrollü Multi-Process tüm senaryolar için istatistiksel sonuçlar 49

SİMGELER VE KISALTMALAR

Simgeler

$\boldsymbol{\theta}$	: Popülasyon Artış Oranı (C-JSO)
$\boldsymbol{\varepsilon}$	: Benzerlik Oranı (C-JSO)
$\boldsymbol{ms}$	: Makespan (tamamlanma süresi)
$\boldsymbol{F^{obj}}$	: Uygunluk Fonksiyonu (Fitness)
$\boldsymbol{e_c}$	**:** Çekiciliği yöneten faktör
$\overrightarrow{\boldsymbol{OC}}$	: Okyanus Akıntısı
φ	: İterasyon değer değişimi takibi
X*	**:** Denizanası şu anda sürüdeki en iyi konuma sahip
$\boldsymbol{\mu}$	**:** Tüm denizanalarının ortalama konumu
$\boldsymbol{U_{b,d}}$	: Arama uzayı üstsınır
$\boldsymbol{L_{b,d}}$	: Arama uzayı altsınır
$\boldsymbol{X_i}$	**:** Denizanası konumunun lojistik kaotik değeri
$\boldsymbol{\alpha}$	**:** Dağılımın standart sapmasıdır
$\boldsymbol{\pm\beta\sigma}$	**:** Her bir denizanasının konuma mesafe olasılığı
$\boldsymbol{\beta}$	**:** Dağıtım katsayısı
$\boldsymbol{\gamma}$	: Hareket katsayısı
$\boldsymbol{c(t)}$	**:** Zaman kontrol fonksiyonu
$\boldsymbol{r}$	**:** Rastgele uniform değer
$\boldsymbol{Df}$	**:** En iyi denizanası konumu ile tüm denizanalarının ortalama konum farkı

Kısaltmalar

C-JSO	**:** Klonlanabilir Denizanası Optimizasyonu (Cloneable Jellyfish Search Optimizer)
PSO	**:** Parçacık Sürü Optimizasyonu (Particle Swarm Optimization)
JSO	**:** Denizanası Arama Optimizasyonu (Jellyfish Search Optimizer)
VM	: Sanal Makine (Virtual Machine)
MIPS	: Sanal Makine İşlem Kapasitesi
MI	: Görev Komut Uzunluğu
ACO	: Karınca Koloni Optimizasyonu (Ant Colony Optimization)
GA	: Genetik Algoritma (Genetic Algorithm)
WOA	: Balina Optimizasyon Algoritması (Whale Optimization Algorithm)
EMA	: Elektromanyetizma Metasezgisel Algoritma
ABC	: Yapay Arı Kolonisi Algoritması (Artificial Bee Colony Algorithm)
DSOS	: Simbiyotik Organizma Arama Optimizasyonu
GWO	: Gri Kurt Optimizasyonu (Grey Wolf Optimization)

1. GİRİŞ

Bulut bilişim, kullanıcıların internet ortamında verilerini depolayabildikleri, fiziksel ve yazılımsal kaynak ihtiyaçlarını kullandığın kadar öde prensibi ile karşılayabilen ölçeklenebilir sistemler bütünüdür. Bulut bilişim kullanıcıları, ihtiyaç duydukları fiziki veya yazılımsal sistemlerin sürekli takibini yapmak yerine ölçeklenebilir uzak bulut sistemler kullanarak kendileri için basit, hızlı ve ekonomik çözümler sağlayabilmektedir. Fiziki sistemlerin korunma, altyapı, bakım-onarım gibi sürdürebilirlik maliyetleri yüksek olduğundan bulut sistemler kullanıcılar için daha ekonomiktir. Bulut sistemler, kullanıcıya kullanım kolaylığı sağlamasının yanında, mevcut kaynakların sınırlı olması, talepleri optimum şekilde karşılayabilmesi, doğal olarak, maliyet ve sürdürülebilirlik açısından hayati öneme sahiptir. Dolayısıyla performans, enerji tüketimi ve sanal makine (VM) paylaşımı gibi sorunlar dikkate alındığında, görev-kaynak dağılımı çizelgelemesi hem servis sağlayıcılar hem de kullanıcılar açısından kritik bir konudur.

Görev çizelgeleme, bulut sistem performansını ve optimum kaynak kullanımını doğrudan etkileyen bir faktördür. İyi tasarlanmamış bir görev veya kaynak çizelgelemesi SLA ihlaline, ciddi maliyet artışına ve performans düşüklüğüne yol açabilir. İş hacmine bağlı olarak müşterilerin sanal makine (VM) sayıları ve özellikleri çeşitlilik gösterebilir. Müşteriler bu sanal makinelerin özelliklerine ve kullanım sürelerine göre bulut sağlayıcıya belirli bir ücret öderler. Sanal makineler üzerinde çalıştırılması gereken görevlerin (Task) yanlış çizelgelenmesi, görev tamamlanma süresinin (makespan) artmasına ve doğal olarak da müşteri için maliyet artışına yol açar. Makespan' deki bu artış dolaylı olarak da bulut sağlayıcının enerji sarfiyatı ve bakım onarım maliyetlerini olumsuz etkiler. Bu nedenle bulut sitemlerde iyi bir görev çizelgeleme algoritmasının kullanılması hem müşteri hem de bulut sağlayıcı açısından zorunludur.

Bulut bilişimde görev çizelgeleme NP-hard problem kategorisinde bulunur [1] ve bu çizelgeleme farklı amaçlara yönelik gerçekleştirilebilir. Bu amaçlar arasında; gecikme (makespan), maliyet, ölçeklenebilirlik, üretim aralığı, güvenilirlik, kullanılabilirlik ve verimlilik gösterilebilir. Bu amaçlardan bir veya birkaçı dikkate alınarak yürütülecek işlev için en uygun kaynakların tahsis edilmesi çizelgeleme problemidir. Çizelgeleme problemlerini çözmede en önemli faktör kullanılacak optimizasyon tekniklerdir. Optimizasyon, belirli sınırlar göz önüne alınarak, bir problemin maksimize veya minimize edilmesidir. Klasik yöntemlerin çözmede yeterli olmadığı durumlarda metasezgisel optimizasyon teknikleri kullanılabilmektedir.

Metasezgisel yöntemler, toplumsal yaşam fenomenlerini esas alarak akıllı arama yapabilen yöntemlerdir [4]. Bu yöntemler, herhangi bir problemin matematiksel modelinin çıkarılamadığı ve kesin çözümün bulunamadığı, optimal çözüme yakın bir çözüm bulmak için kullanılmaktadır ve hesaplama açısından genellikle avantajlıdır. Deterministtik çözümlerin yerine metasezgisel algoritmaların kullanılması, performans açısından görev çizelgeleme problemlerinin çözümünde

sıklıkla tercih edilmektedir [2]. Öte yandan problem türünden dolayı rastgele arama temelli metasezgisel algoritmaların lokal minimumlara takılma olasılıkları oldukça yüksektir [4]. Bu olasılık sanal makine ve görevlerin artmasıyla daha çok artabilmektedir. Bu sebeple kullanılan metasezgisel algoritmaların bu sorunu aşacak mekanizmaları kullanması gereklidir. Bu tez çalışmasında, söz konusu problemin çözümü için farklı bir yaklaşım mekanizması kullanan ve güncel sürü tabanlı metasezgisel bir algoritma olan Denizanası Arama Optimizasyonu (Jellyfish Search Optimizer-JSO*)* temelli bir çözüm önermektedir [3]. Önerilen yöntemin en özgün yanı, farklı bir benzerlik kontrolü ile dinamik popülasyon artışına imkân vermesi ve böylece algoritmanın lokal minimumlardan daha hızlı bir şekilde kurtulmasına olanak sağlamasıdır. Bu yöntem ile bulut sistemlerde kullanıcıdan gelen isteklerin yoğunluğuna göre kaynak ayrılması ve gereksiz kaynak kullanımının önüne geçilmesine yardımcı olunabilecektir.

Klasik Denizanası Arama algoritması, okyanusta ki denizanalarının yiyecek bulmak gibi temel ihtiyaçlarını giderirken sergiledikleri hareketlerin, sürüdeki diğer bireyleri nasıl etkilediğini ve sürünün amacına daha kolay ulaşmak için davranışlarından esinlenerek geliştirilmiş bir optimizasyon algoritmasıdır [3]. Önerilen Klonlanabilir Denizanası Algoritması (C-JSO), klasik algoritmasından farklı olarak sabit popülasyon sayısı yerine içinde bulundurduğu mekanizmalar sayesinde çalışma anında dinamik olarak popülasyon artışı yapabilmektedir. Dolayısıyla sabit popülasyonda arama uzayının belirli bir alanında çalışma yapmak yerine algoritmanın ihtiyacına göre alternatif adaylar ile dinamik bir popülasyon kullanmak esnekliği arttıracaktır.

Bulut ortamlarda görev çizelgeleme için optimum çözümleri bulmak, araştırmacıların son zamanlarda ilgilendikleri önemli bir konudur. Tsai vd. [5] bulut bilişim sistemleri için daha iyi çizelgeleme çözümleri bulmak için Genetik Algoritma (GA), Karınca Koloni Algoritması (ACO) ve Parçacık Sürü Optimizasyonunu (PSO) tek bir çerçeveye (Framework) entegre ederek Hiper Sezgisel Çizelgeleme Algoritması (HHSA) önermişlerdir. Önerilen yöntemin performansını değerlendirmek için, CloudSim ve Hadoop üzerinde detaylı testler gerçekleştirilmiştir. HHSA hem CloudSim hem de Hadoop gibi zamanlama algoritmalarıyla karşılaştırıldığında, görev zamanlamasının tamamlanma süresini önemli ölçüde azaltabileceğini göstermiştir. Baykan vd. [6] bu problemin çözümü için parçacık sürüsü optimizasyonuna (PSO) ve karınca kolonisi optimizasyonuna (ACO) dayalı yeni bir hibrit algoritma sunmaktadır. Bu hibrit çalışmayı karınca parçacık optimizasyon algoritması (HAP) olarak adlandırmaktadır. Önerilen yöntemde, ACO ve PSO her yinelemede ayrı ayrı çalışır ve çözümlerini üretir. En iyi çözüm, sistemin en iyisi olarak seçilir ve parametreleri, bir sonraki yinelemede parçacıkların (karıncaların) yeni konumunu seçmek için kullanılır. Önerilen yöntemin performansı kıyaslama problemlerinde PSO ve ACO ile karşılaştırılmış, sonuç olarak HAP algoritması ile daha kaliteli sonuçlar elde edilmiştir.

Awad vd. [7] güvenilirliği dikkate alan bulut bilişim için Yük Dengeleme Mutasyonu (dengeleme) ve parçacık sürüsü optimizasyonu (LBMPSO) tabanlı çizelgeleme tekniği kullanarak

matematiksel model önermiştir. Yürütme süresi, iletim süresi, gidiş-dönüş süresi, iletim maliyeti ve sanal makine arasındaki yük dengeleme dikkate alınmıştır. LBMPSO, mevcut kaynakları dikkate alarak bulut bilişim ortamının güvenilirliğine ulaşmada rol oynayabilir ve tahsis edilemeyen görevi yeniden planlayabilir. LBMPSO, standart PSO, rastgele algoritma ve En Uzun Buluttan En Hızlı İşlemciye (LCFP) algoritması ile karşılaştırılmış, sonuç olarak üretim süresi, yürütme süresi, gidiş-dönüş süresi ve iletim maliyetinde tasarrufun önerilen yöntemle sağlanabileceği gösterilmiştir.

Alsaidy vd. [8] çeşitli teknikler kullanarak parçacık sürüsü optimizasyonunun (PSO) geliştirilmiş bir versiyonunu önermektedir. PSO' yu başlatmak için En Uzun Buluttan En Hızlı İşlemciye (LCFP) ve minimum tamamlama süresi (MCT) algoritmaları kullanılmıştır. Önerilen LJFP-PSO ve MCT-PSO algoritmalarının performansı, üretim süresi, toplam yürütme süresi, dengesizlik derecesi ve toplam enerji tüketimi optimizasyonun amaçlarıdır. Simülasyon sonuçları, önerilen LJFP-PSO ve MCT-PSO' nun diğer algoritmalara kıyasla etkinliğini anlatmışlardır. Yıldırım vd. [9] yüksek kaliteli kural setinin otomatik madenciliği, ihtiyaçların doğası gereği çok amaçlı bir optimizasyon problemi olarak ele alınmış ve bu görev için Gray Wolf Optimizer tabanlı yeni uyarlanabilir çok amaçlı akıllı arama ve optimizasyon algoritmaları önerilmiştir. Bir diğer çalışmada Yıldırım vd. [10] Hadoop temelli yumuşak hesaplama algoritmalarının çalıştırılması üzerine deneysel bir çalışmanın sonuçlarını sunmaktadırlar. Bu çalışma, Hadoop üzerinde çalışan basit bir genetik algoritmanın yüksek boyutlu optimizasyon problemlerine çözüm üretmek için nasıl kullanılabileceğini göstermektedir. Ayrıca MapReduce zincirlerine ihtiyaç duymayan basit ama etkili bir teknik önerilmiştir.

Zhao vd. [11] bulut sistemlerdeki görev çizelgeleme problemini çözmek için meta-sezgisel bir algoritma önermiştir. Bu çalışma, var olan Karınca Kolonisi Optimizasyonunu (ACO) geliştirerek Yük Dengeleme Karınca Kolonisi Optimizasyonu (LBACO) algoritmasına dayalı bir bulut görev zamanlama politikası benimsemiştir. Çalışmanın ana katkısı, belirli bir görev setinin yapım süresini en aza indirmeye çalışırken tüm sistem yükünü dengelemektir. Belgacem vd. [12] görev zamanlaması, kaynak tahsisinin kritik bir parçası olduğunu ve Elektromanyetizma Meta-sezgisel Algoritması (EMA) temelli çözüm önermişlerdir. EMA' yı, sanal makinelerde (VM) yürütme görevlerinin süresini en aza indirecek şekilde özel olarak tasarlanmasını, kaynak kullanılabilirliği üzerindeki etkisini de incelemişlerdir. Simülasyon sonuçları, EMA' nın PSO' ya daha yakın uygun sonuçlar verdiğini belirtmişlerdir. Yıldırım vd. [13] son yılların başarılı optimizasyon algoritmalarından olan Ayçiçeği Optimizasyon algoritması ile optimizasyon temelli yaklaşım kullanarak veri setleri üzerinde herhangi bir ön işlem yapmaya gerek kalmadan açıklanabilir kurallar üretilebileceğinin ispatını göstermişlerdir.

Sasikaladevi [14] MMSF adında bir minimum makespan görev çizelgeleme çerçevesi ve MMA adında bir minimum makespan görev çizelgeleme algoritması önermektedir. Bu algoritma

iki amaç için geliştirilmiştir; bunlar toplam üretim süresini en aza indirmek ve sanal makine kullanımını en üst düzeye çıkarmaktır. Görev çizelgeleme problemi çok amaçlı optimizasyon problemi olarak formüle edilmiştir. Deneyler, MMA algoritmasının sanal makine kullanımına dayalı geleneksel görev zamanlama algoritmalarından daha iyi performans gösterdiğini iddia etmiştir. Liu vd. [15] bulut ortamlarında görev çizelgelemenin özelliklerini hedefleyen, genetik-karınca kolonisi algoritmasına dayalı bir yaklaşım önermektedir. Algoritmanın yakınsama hızı üzerinde karınca kolonisi optimizasyonunun (ACO) güçlü pozitif geri bildiriminden yararlanılmaktadır. Algoritma, optimal çözümü hızlı bir şekilde çözmek için genetik algoritmanın global arama yeteneğini kullanır ve ardından bunu ACO' nun ilk feromonuna dönüştürür. Simülasyon deneyleri, aynı koşullar altında, bu algoritmanın genetik algoritmaya ve ACO' ya ağır bastığını, hatta büyük ölçekli ortamlarda verimlilik avantajına sahip olduğunu göstermektedir.

Calheiros vd. [16] uygulama iş yükü modellerinin ve kaynak modellerinin performansını, değişen sistem ve kullanıcı gereksinimlerinin takibini yapmak için CloudSim' i geliştirmişlerdir. CloudSim araç seti, veri merkezleri, sanal makineler (VM) ve kaynak sağlama ilkeleri gibi Bulut sistem bileşenlerinin hem sistem hem de davranış modellemesini destekler. CloudSim' in kullanışlılığı, hibrit birleşik bulut ortamında uygulama hizmetlerinin dinamik olarak sağlanmasını içeren bir vaka çalışmasıyla gösterilmiştir. Bu vaka çalışmasının sonucu, Bulut bilgi işlem modelinin, değişken kaynak ve hizmet talep modelleri altında uygulama QoS gereksinimlerini önemli ölçüde iyileştirdiğini savunmuşlardır. Abdullahi vd. [17] bulut kaynaklarında görevlerin optimal zamanlaması için bir Ayrık Simbiyotik Organizma Arama (DSOS) algoritması önermişlerdir. Simbiyotik Organizma Arama (SOS), sayısal optimizasyon problemlerini çözmek için yeni geliştirilmiş bir meta-sezgisel optimizasyon tekniğidir. SOS, bir ekosistemdeki organizmalar tarafından sergilenen simbiyotik ilişkileri taklit eder. Simülasyon sonuçları, DSOS' un görev çizelgeleme problemlerinde kullanılan en popüler sezgisel optimizasyon tekniklerinden biri olan Parçacık Sürü Optimizasyonundan (PSO) daha iyi performans gösterdiğini savunmuşlardır. DSOS, arama büyüdüğünde daha hızlı yakınsar ve bu da onu büyük ölçekli zamanlama sorunları için uygun hale getirir.

Chen vd. [18] ilk kez, verilen bilgi işlem kaynaklarıyla ve çok amaçlı bir optimizasyon modeliyle bulut görev zamanlaması için metasezgisel balina optimizasyon algoritmasını (WOA) önermişlerdir. Bu temelde, WOA tabanlı yöntemin optimal çözüm arama kabiliyetini daha da geliştirmek için IWC adlı gelişmiş bir yaklaşım kullanılmaktadır. Li vd. [40] bulut bilişim sistemindeki esnek görev çizelgeleme problemi için hibrit ayrık yapay arı kolonisi (ABC) algoritması önermişlerdir. Yöntemde hem tek bir amaç hem de birden çok amaç göz önünde bulundurulabilmektedir. Çok amaçlı HFS problemlerinde, maksimum tamamlama süresinin, maksimum cihaz iş yükünün ve tüm cihazların toplam iş yüklerinin minimizasyonu gibi üç hedef

aynı anda ele alınır. Aynı paralel makinelere sahip HFS ve ilgisiz makinelere sahip HFS olmak üzere iki farklı HFS türü göz önünde bulundurulur. Önerilen algoritmada klasik ABC algoritmasında olduğu gibi işçi arı, gözcü arı ve kâşif arı olmak üzere üç tür yapay arı yer almaktadır. Benzer şekilde, Maheswari vd. [41] ABPS algoritması olarak adlandırılan yapay arı kolonisi algoritması ve parçacık sürüsü optimizasyonunun birleştirilmesiyle hibrit bir algoritma tanıtmıştır. Önerilen ABPS algoritması, minimum üretim süresi, maliyet, maksimum kaynak kullanımı ve yük dengelemesi ile bulut ortamında görev çizelgelemeyi optimize eder.

Pirozmand vd. [42] bulut bilişimde Görev Zamanlama Problemini çözmek için GSAGA adlı hibrit bir algoritma sunmaktadır. Çalışmada, Genetik Algoritma' nın (GA), kararlılık ve yerel arama açısından zayıf bir performans sergilediği iddia edilmiş ve bu nedenle GA' nın genel arama kapasitesini Yerçekimi Arama Algoritması (GSA) ile birleştirerek yeni bir hibrit yaklaşım denenmiştir. Deneysel sonuçlar, önerilen algoritmanın problemi en son teknolojiye kıyasla daha yüksek verimlilikle çözebileceğini göstermiştir. Babu vd. [43] bulut sistem genelinde yükü dengelemek için Hibrit Parçacık Sürü Optimizasyonu (HPSO) tabanlı bir çizelgeleme yöntemi önermektedir. Deney sonuçları, hibrit PSO' nun mevcut basit PSO algoritmasını kullanmaya kıyasla daha etkili olduğunu savunmuşlardır. Benlalia vd. [44] ise görev çizelgeleme problemlerini çözmek için Tabu Search (TS) ile birleştirilmiş Kedi Sürü Optimizasyonu (CSO) önermektedir. Önerilen hibrit algoritma CloudSim üzerinde test edilmiş ve önerilen algoritmanın performansı, PSO ve FCFS' nin performansıyla karşılaştırılmıştır.

Saif vd. [45] kullanıcıların görevlerini birden çok hesaplama kaynağına atayan bağımsız bir görev zamanlama algoritması tanıtmaktadır. Önerilen algoritma, genetik algoritma (GA) ve parçacık sürüsü optimizasyonuna (PSO) dayalı hibrit bir algoritmadır. Simülasyon sonuçları, önerilen algoritmanın, işlem süresini azaltmada ve kaynak kullanımını arttırmada GA ve PSO algoritmalarından daha iyi performans gösterdiğini ispatlamıştır. Safi vd. [46] balinaların gruplandırılmasında yeni bir fikir sunarak Balina optimizasyon algoritmasının geliştirilmiş GWOA adı verilen bir versiyonunu tanıtmıştır. Önerilen yöntemde, ilk olarak erken yakınsama sorununun üstesinden gelinmesi ve daha sonra optimal çözümün bulunmasında yerel ve küresel arama arasında bir denge kurulması önerilmektedir. Önerilen yöntem, sıralanmış popülasyonu gruplara böler ve aramayı özelleştirir. Bir sonraki adımda, GWOA, ortalama yürütme süresini, yanıt süresini azaltmak ve verimi artırmak için yüksek iş yükünde bir bulut bilişim zamanlayıcısında kullanılır. Önerilen yöntemin performansı, standart balina optimizasyon algoritması (WOA), geliştirilmiş balina optimizasyon algoritması (CWOA), parçacık sürü optimizasyonu (PSO) ve yarasa algoritmaları sonuçlarına göre incelenmiştir.

Choe vd. [47] amaca göre düzenlenmiş bir Karınca Kolonisi Optimizasyonu kullanan bir görev zamanlama algoritması önermektedir. Önerilmiş versiyon, karıncaların hedef makineye karar vermesi için istatiksel bir yaklaşım kullanmaktadır. Deneysel sonuçlar, önerilen algoritmanın

ortalama yürütüm süresini yaklaşık %6,4'e düşürdüğünü göstermektedir. Malekloo vd. [48] Çok Amaçlı Karınca Koloni Optimizasyonu (MACO) temelli yerleştirme algoritmasını sunmuşlardır. Bu algoritmayla toplam kaynak tüketimi, güç tüketimi ve ağ elemanları arasındaki iletişim masraflarını en aza indirerek sanal makine yerleştirme problemini çözmek amaçlanmıştır. MACO algoritması Cloudsim benzetim ortamında Mikro Genetik Algoritması (MGA), Lokal Regresyon (LR), Dinamik Voltaj ve Frekans Ölçeklendirme (DVFS) ve FFD algoritmalarıyla karşılaştırılmıştır. Sonuçta MACO' nun enerji tüketimi ve iletişim enerjisi maliyeti konusunda daha başarılı olduğu görülmüştür.

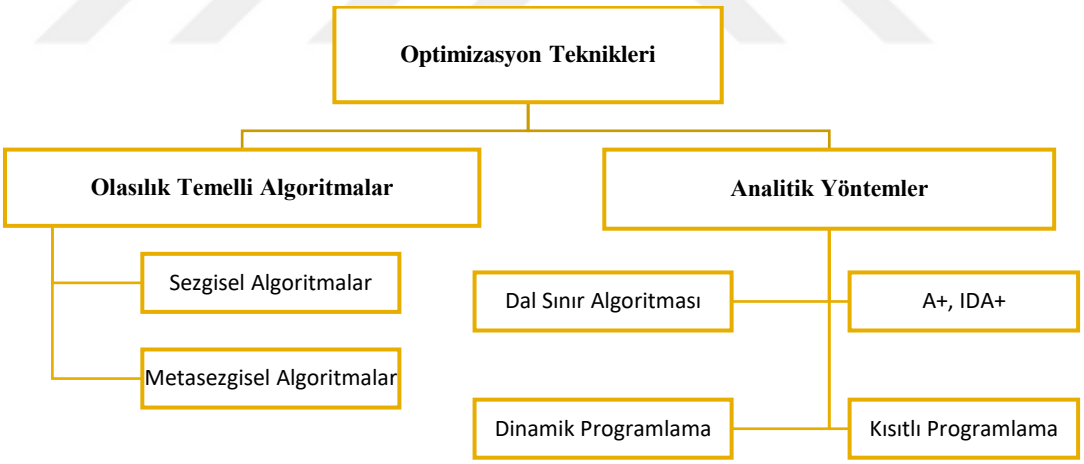
Genel olarak literatürde önerilen metasezgisel yaklaşımlar, iterasyonlar esnasında ortaya çıkabilen lokal minimalara takılma problemini daha kısa sürede aşma noktasına odaklanmamışlardır. Her ne kadar amaç temelde aynı olsa da metasezgisel yöntemlerin lokal minimaya takılma handikabı, bulut sistemlerde görev çizelgeleme gibi karmaşık problemler için dikkate alınmalıdır. Ayrıca yazarın bildiği kadarıyla bu problem tipi için metasezgisel yaklaşımlarının paralel versiyonları denenmemiştir. Bu amaçla tez çalışmasında iki yaklaşım ele alınmış ve sorgulanmıştır. Birincisi, literatürdeki örneklerinden farklı olarak durum kontrollü dinamik popülasyon kullanımı ile lokal minimalardan kurtularak mevcut çözümlere daha hızlı ulaşmak mümkün müdür? İkincisi ise metasezgisel algoritmanın paralel yürütümü başarımı artırabilir mi?

2. TEMEL KAVRAMLAR VE KAPSAM

Son yıllarda teknolojinin ilerlemesiyle birlikte kullanıcıların sanal dünyada artan taleplerini karşılayabilmek için çeşitli teknikler geliştirilmiştir. Optimizasyon tekniği, ekonomik açıdan sağladığı kazançların yanında kullanıcı ve sağlayıcı arasındaki etkileşimi de arttırmıştır. Ayrıca sağladığı kolaylıklarla beraber sistemde yer alan kaynakların kalitesinin yükseltilmesinde de etkin bir araç olarak kullanılmaktadır. Bu bölümde optimizasyon, metasezgisel yaklaşımlar ve bulut sistemler hakkında bilinmesi gereken temel bilgiler paylaşılacaktır.

2.1. Optimizasyon

Optimizasyon, istenilen amaç veya amaçlar doğrultusunda belirli kısıtlamalara tabi tutularak en uygun çözümün elde edilme süreci olarak tanımlanabilir. Özetle kısıtlı alanlar içerisinde istenilen amaç/amaçlar için en iyi olanı aramaktır. Bulunan en iyi çözüm optimum olarak adlandırılır [19]. Optimizasyon, gelen taleplere karşı karar verme süreçlerini hızlandırmak ve karar kalitesini arttırmak için kullanılır. Optimizasyon, maliyet açısından getirdiği kazançların yanında kullanıcı ve sağlayıcı arasındaki koordinasyonu sağlaması için sıkça başvurulan bir yöntemdir.



Şekil 2.1. Optimizasyon Teknikleri

Optimizasyon problemlerinin çözümünde klasik yöntemler olarak adlandırılan matematiksel yöntemler önceleri çok yaygın olarak kullanılmaktaydı. Bu tür yöntemlerin esnek olmaması ve matematiksel fonksiyonlarla tanımlama gereksinimi gibi dezavantajları, son zamanlarda, araştırmacılarda genel amaçlı ve performansı yüksek yöntemler geliştirme çabalarını artırmıştır. Bu çabalar doğrultusunda Şekil 2.1'deki gibi kategorize edilebilecek optimizasyon türleri çıkmıştır [24]. Bu kategori yapısına göre bu yöntemler ile ilgili temel bilgiler aşağıda verilmiştir.

2.1.1. Analitik Algoritmalar

Belirli toleranslar sağlandığında kesin çözümlerin alındığı matematiksel temelli yöntemlerdir. Olasılık temelli algoritmalara nazaran daha küçük problemlerin çözümünde kullanılırlar. Büyük boyutlu problemlerde olasılık temelli algoritmaların tersine, sonuca ulaşmanın çok uzun sürdüğü veya ulaşamadığı yöntemlerdir.

2.1.2. Sezgisel Algoritmalar

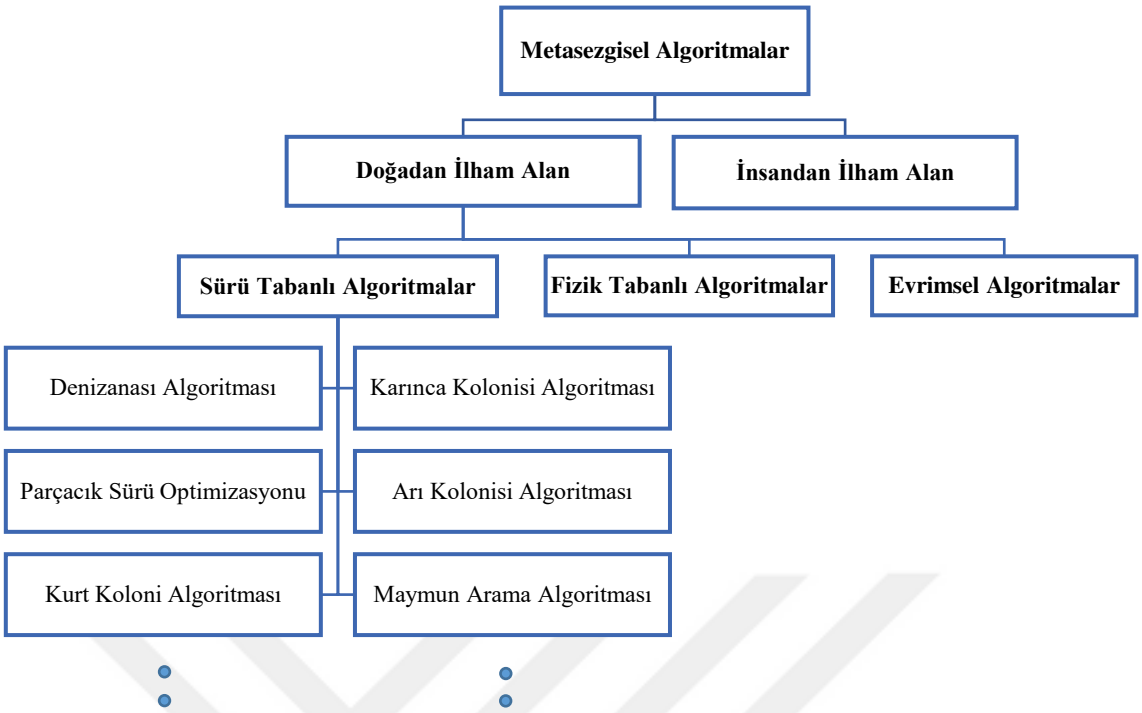
Problem çözmeye dayalı ve belirli bir algoritmaya bağlı kalan yöntemlerdir. Sezgisel ve Metasezgisel algoritmalar, belirli sınırlar içerisinde olasılık temelli arama gerçekleştiren yöntemlerdir [20]. Tahmine dayalı optimum çözümü garanti etmemekle birlikte klasik yöntemlere göre daha hızlı çözüm üretirler. Bu optimizasyon problemlerinde optimuma en yakın sonucu bulmayı hedeflemektedir. Bu yöntemler her adımda oluşturulan çözüm kümesinden yola çıkarak yeni çözümler üretmektedirler.

Problem, optimum çözümü bulunamayacak kadar karmaşıksa, sezgisel yöntemler sezgiye veya bazı deneysel kayıtlara dayanan karar kuralları ile belirli sayıda adımdan sonra en iyi olmasa da tatminkâr bir sonuç verirler. Sezgisel algoritmalar en iyi sonucu garanti etmemekle beraber, kısa sürede optimale yakın sonuçlar verebilir [21].

2.1.3. Metasezgisel Yöntemler

Teknik olarak ilk defa 1986 yılında dile getirilen metasezgisel terimi Yunanca “meta” kelimesi ile “heuristic” kelimesinin birleşiminden meydana gelmektedir ve “daha ileri sezgisel” veya “üst seviye sezgisel” şeklinde ifade edilmektedir [22]. Metasezgisel optimizasyon algoritmaları; basit kavramlara dayanırlar, uygulanması kolaydır, amaç işlevinin eğimi hakkında bilgi gerektirmezler, yerel minimumları atlayabilirler ve çeşitli alanlarda çok çeşitli problemleri çözmek için kullanılabilirler [6].

Kesin sonuçların alındığı yöntemlerden farklı olarak metasezgisel yaklaşımlar, optimum yerine optimuma yakın çözümler elde edilmesini sağlar [23]. Bir problem türü için geliştirilen sezgisel bir yöntemin başka bir problem türüne uygulanması zordur. Ancak son yıllarda problem özelliklerinden bağımsız, genel olarak tanımlanmış ve metasezgisel olarak adlandırılan yöntemlere ilgi artmıştır [21]. Metasezgisel algoritmalar, belirlenen sınırlar içerisinde tahmine dayalı mantıklı arama gerçekleştirerek arama uzayında en optimuma yakın noktalarda arama yapar ve lokal minimumlardan kurtularak en uygun çözüme ulaşmaya çalışır.



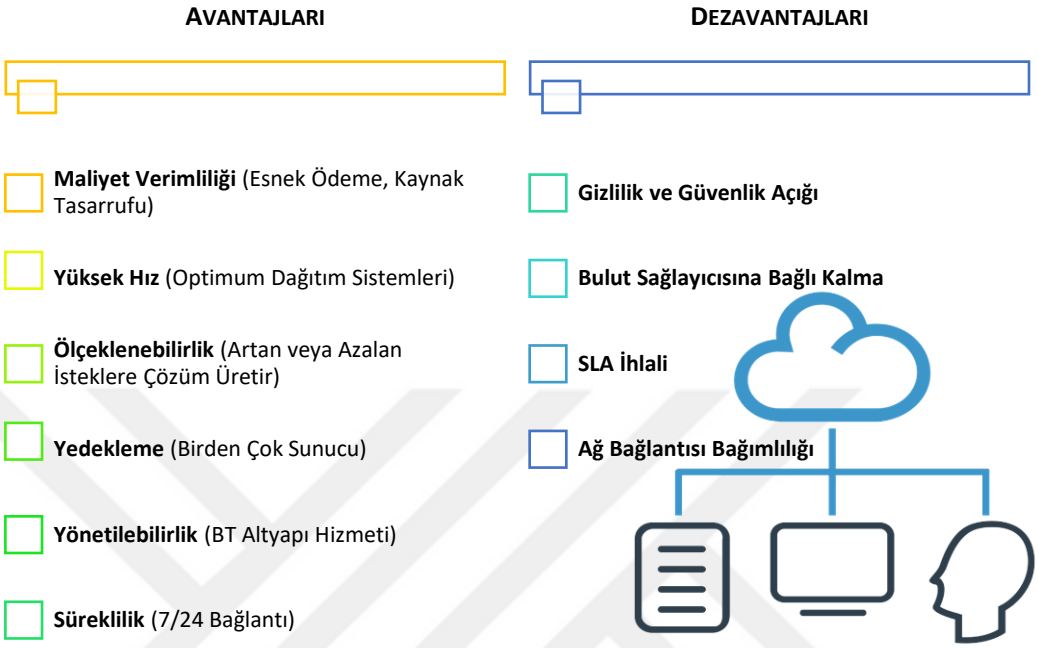
Şekil 2.2. Metasezgisel Algoritmaların Sınıflandırılması

Literatürde çok farklı metasezgisel optimizasyon yaklaşımı önerilmiştir. Bunların hepsinden bahsetmek bu tez çalışmasının kapsamı dışındadır. Literatürde önerilmiş metasezgisel algoritmalar Şekil 2.2'de gösterildiği gibi sınıflandırılabilir. Bu tez çalışmasında, önemli bir metasezgisel yaklaşım olan sürü tabanlı algoritmaların kullanılması tercih edilmiştir. Sürü tabanlı metasezgisel algoritmalar, sürü halinde hareket eden hayvanların yiyecek bulmak gibi temel ihtiyaçlarından esinlenerek geliştirilmiş optimizasyon yöntemidir. Birçok sürü tabanlı optimizasyon algoritması bulunmakla beraber bunlar arasında, Parçacık Sürü Optimizasyonu, Denizanası Arama Algoritması, Karınca Koloni Optimizasyonu, Ateşböceği Sürü Optimizasyonu, Maymun Arama Optimizasyonu, Gri Kurt Algoritması genellikle ilk akla gelenler arasındadır.

2.2. Bulut Bilişim Sistemleri

Bulut teknolojisi, internete bağlanabilecek tüm cihazlar ile uzak sunucular arasında sürekli veri, yazılım, platform ve kaynak paylaşımına imkân sağlayan bununla birlikte muazzam büyük altyapı desteği sunan bilişim sistemleridir. Günümüz bulut bilişim hizmetleri müşterilerine ölçeklenebilir seçenekler ve kullandığın kadar öde sistemi ile ekonomik çözümler sağlayabilir [25]. Fiziki sistemlerin kurulum, maliyet, korunma, altyapı, bakım ve sürekli takibini yapmak yerine 'kullandıkça-öde' mantığına dayalı bu online depolama hizmeti kullanıcılara oldukça kolaylık sağlamaktadır [26]. Bulut bilişim, kuruluşlar açısından kullanıcıdan gelen talepleri en optimum

şekilde dağıtımını yapan mekanizmalara sahiptir. Şekil 2.3' te bulut bilişimin kullanıcılara ne gibi avantaj veya dezavantaj sağladığı görülebilir.

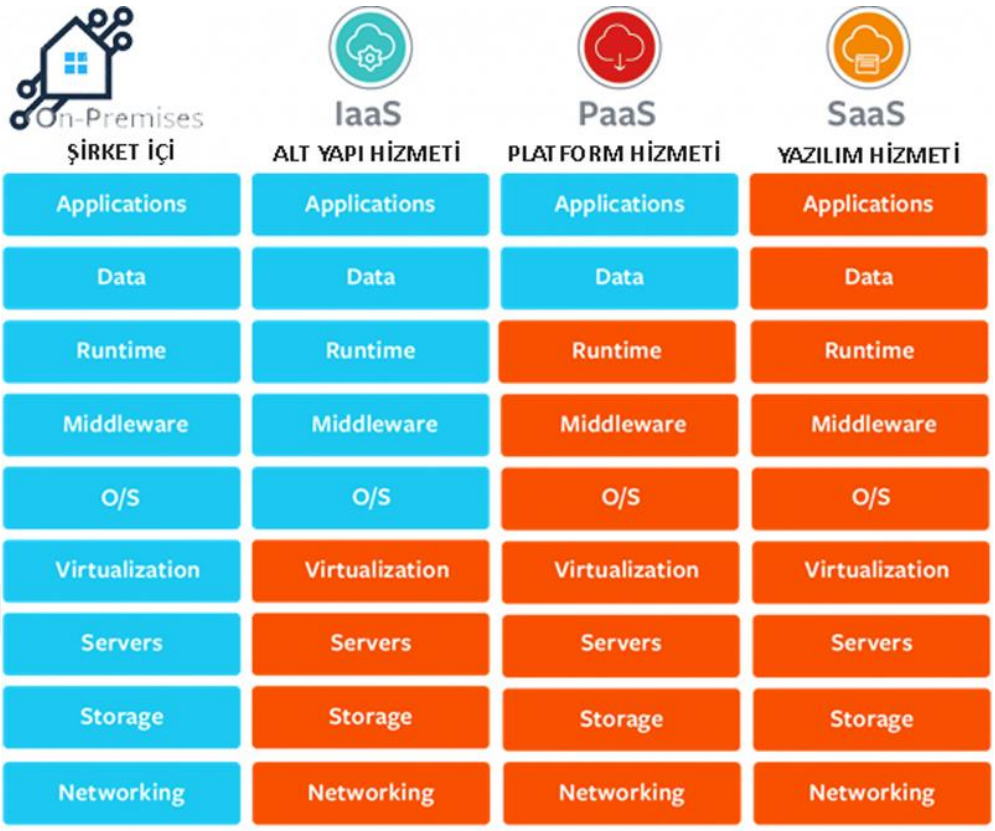


Şekil 2 3. Bulut Bilişim Sistemlerinin Ayrıcalıkları

2.2.1. Bulut Bilişim Hizmet Modelleri

Bulut bilişim sağlayıcıları üç temel modele göre hizmetlerini sunarlar [27]. Bu modeller kullanıcının ihtiyacına göre geliştirilmiş yapılardır. Kullanıcılar bulut sağlayıcısından yalnızca yazılım veya hem yazılım hem donanım hizmeti satın alabilir. Şekil 2.4' te bulut bilişim hizmet modelleri verilmiştir. Burada Altyapı hizmeti (IaaS), Yazılım hizmeti (SaaS), Platform hizmeti (PaaS) ve Şirket içi (On-Premises) farkları açıkça belirtilmiştir. Şirket içi (On-Premises), kuruluşların yazılım, altyapı, sunucular, uygulama ve yürütme hizmetlerini kendi içerisinde barındırdığı hizmetlerdir. Bu hizmetlerin bir kısmı veya birkaçını kullanmayı tercih eden kuruluşlar bulut hizmet sağlayıcılarından destek almaktadır.

a) **Altyapı Hizmeti (IaaS):** Bulut bilişimdeki hizmet modellerinin temelini oluşturan bu hizmet, genel olarak bakıldığında donanım altyapısı sunar. Sunucular, sanal makinalar ve ağ altyapısı hizmetler sağlanır. Burada sanal makinaların özellikleri kullanıcı tarafından belirlenir ve ölçeklendirme bulut sağlayıcılar tarafından yapılır. Burada donanım altyapısının yönetilmesi sağlayıcıya aittir ve yazılım işlerinin yönetilmesi kullanıcı tarafından yapılır.



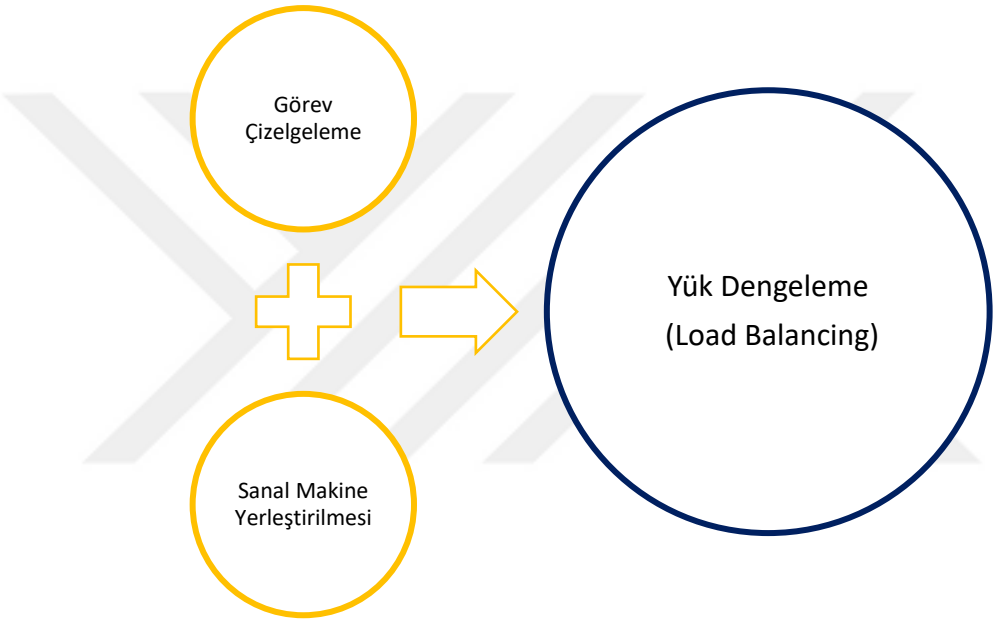
Şekil 2.4. Bulut Bilişim Sağlayıcıları Hizmet Modelleri

b) **Platform Hizmeti (PaaS):** Bulut sağlayıcılar bu hizmet modelinde, kullanıcılara yazılım olarak kısmi hizmet vermektedir. Bu platformda, altyapı hizmetlerine ek olarak işletim sistemi, programlama dili, yürütme ortamı ve veritabanı gibi hizmetler sunmaktadır. Bu hizmetler sağlayıcı tarafından karşılanır ve kullanıcı müdahalede bulunamaz. Burada kullanıcı, oluşturduğu uygulama ve verileri bu sisteme taşımaktadır. Kullanıcı temel bulut altyapılarını yönetemez ve bu altyapılar üzerine yerleştirilen uygulamaların depolama ve spesifik ayarları üzerinde yetkileri bulunur.

c) **Yazılım Hizmeti (SaaS):** Bu modelde bulut sağlayıcılar, kullanıcılar için yazılım ve altyapı arasındaki yönetimi de sağlarlar. Burada kullanıcıların altyapı üzerinde değişiklik yapma yetkisi yoktur ve sadece basit yazılım işlemleri kullanıcı tarafından sağlanır. Esneklik, çalışma süresi zarfında değişen iş taleplerini karşılamak için görevlerin çoklu sanal makinelerde klonlanmasıyla sağlanır [28]. Bu modelde, yük dengeleyiciler kullanıcıdan gelen talepleri sanal makinelere dağıtımını yapar. Bu işlemleri yüzeysel görme yetkisine sahip bulut sistem kullanıcıları fark edemezler. Ayrıca bulut sistem kullanıcıları donanım ve yazılım güncellemelerinden sorumlu tutulamaz.

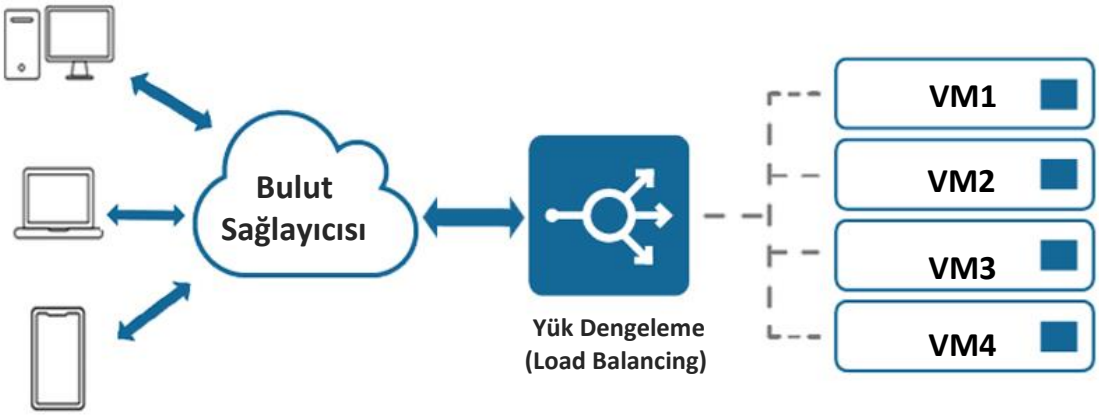
2.2.2. Yük Dengeleyici (Load Balancing)

Fiziksel makineler üzerinde bulunan farklı sayılardaki sanal makinelerin değişken olabilen farklı miktardaki kaynak gereksinimleri, başarımda ve hizmet kalitesinde düşüşe neden olabilmektedir. Bu sıkıntıların çözümü için yük dengelemenin yapılması gerekmektedir [37]. Yük dengeleme, bulut ortamlarda donanım/yazılım trafiğinin kurallı ve verimli dağıtımı olarak tanımlanır. Yük dengeleyiciler, sunucular ile kullanıcılar arasında konumlanır ve gelen talepleri sunuculara uygun şekilde dağıtır. Şekil 2.5' te yük dengeleme ve görev çizelgeleme ilişkisine bir örnek gösterilmektedir [38].



Şekil 2.5. Yük dengeleme çeşitleri

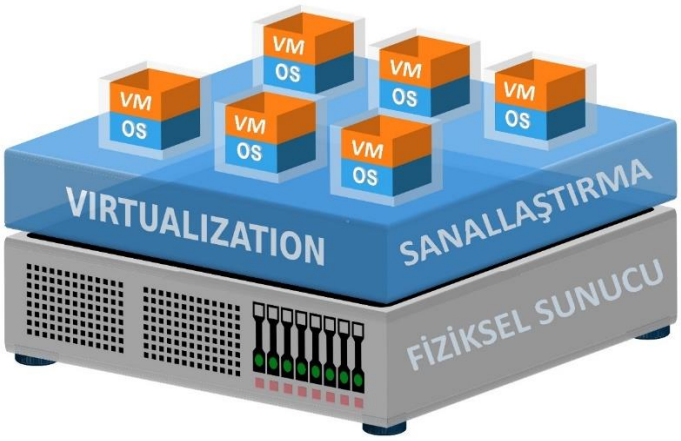
Yük Dengeleme, çok uygulamalı ve cihazlı iş akışlarında, gelen sayısız talebin yönetilebilmesi için en uygun metodolojidir. Gelen talepleri anlık ve sistematik bir şekilde doğru konumlara yönlendirme görevini üstlenir. Bu sayede yüksek maliyetlere sebebiyet verebilecek tıkanıklıkları ve öngörülemeyen sorunların önlenmesini sağlar. Yalnızca karmaşık BT ortamlarında yönetici ve yönlendirici rolü üstlenmez; aynı zamanda gerekli performansı ve güvenliği de sağlar. Şekil 2.6' da yük dengelemenin bulut ortamlardaki konumu gösterilmiştir.



Şekil 2.6. Bulut sistemlerde yük dengeleme

2.2.3. Sanallaştırma

Sanallaştırma, fiziki bir sistem içerisinde birden çok sanal makine (VM) oluşturularak gelen görevleri (Task) aynı anda birkaç makinede hızlı ve esnek bir şekilde cevaplamasını sağlayan bulut bilişim mekanizmasıdır. Sanallaştırmanın amacı, gelen talepleri karşılamak için ayrı fiziki sistemler almak yerine bir sistemin içerisine sanal sistemler kurarak maliyet tasarrufu kazanmaktır. Sanallaştırma aynı zamanda depolama ve ağ altyapılarında da sıkça kullanılan bir tekniktir. Bulut ortamlarda sanallaştırmanın artılarından biri de kullanıcının ihtiyacına göre sanal makine oluşturabilmesidir. Kullanıcı sanal makine oluştururken işlemci (CPU), bellek (RAM) ve depolama gibi özellikleri seçebilir. Seçtiği bu özellikler yetersiz gelmesi durumunda sağlayıcıdan herhangi bir özelliğin yükseltilmesini talep edebilir. Farklı bulut bölgelerindeki sunuculara verileri yedekleyebilir ve bu imkân güvenlik açısından oldukça önemlidir. Şekil 2.7 ‘de bir sunucu üzerine kurulmuş birden çok sanal makine (VM) ve birbirinden bağımsız çalışan işletim sistemleri (OS) temsili olarak gösterilmiştir.



Şekil 2.7. Fiziksel sunucu üzerindeki sanal makinenelerin temsili gösterimi

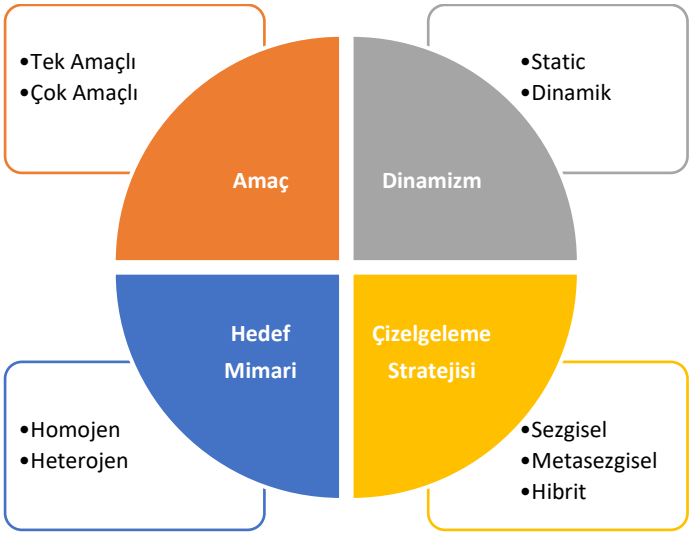
Bulut sistemleri donanım sanallaştırması, programlama dili sanallaştırması, sunucu konsolidasyonu ve sanal makine yer değişimi tekniklerini kullanır. Donanım sanallaştırması IaaS' deki çözümlere uygulanır. Programlama dili sanallaştırması ise PaaS' de kullanılan bir teknolojidir. Sunucu konsolidasyonu ve sanal makine yer değişimi, bilişim hizmeti verirken izolasyon ve yönetilebilirliği sağlamak için uygulanır [29]. Dağıtılmış bilgi işlem bileşenlerine büyük bir veri ortamında erişmek, depolamak, analiz etmek ve yönetmek için gereken birçok platform sanallaştırma yoluyla sağlanmaktadır [30]. Bulut ortamlarda oluşturulan sanal makinelere görev tahsisini yapan mekanizmaya görev planlama (çizelgeleme) denir.

2.2.4. Görev Çizelgeleme (Task Scheduling)

Bulut ortamlarda kaynaklar sınırlı olduğu için güç tasarrufu ve maliyet açısından görev-kaynak dağılımı planlamasına (çizelgeleme) ihtiyaç duyulmaktadır. Görev çizelgeleme, kullanıcıdan gelen talepleri en optimum şekilde kaynaklara dağılımını yapan bulut sistem mekanizmasıdır. Görev-kaynak tahsisi iyi tasarlanmadığında SLA ihlaline, maliyet artışına ve performans düşüklüğüne yol açabilir. Kullanıcıdan gelen sayısız görev sayıları ve bunların birçok sanal makineye optimum şekilde dağıtımı dikkate alındığında görev çizelgelemenin ne denli zor olduğu daha net anlaşılabilir.

Görev çizelgeleme çalışma prensibine bakıldığında, önceden tanımlanmış n tane görevin (Task) bir veya daha çok amaca göre m adet sanal makineye (VM) en optimum şekilde atanması olarak ifade edilebilir [1]. Bulut sistemlerde sanallaştırma teknikleri ile fiziksel sunucular üzerinde farklı özelliklere sahip sanal makineler (VM) oluşturulabilir. VM' ler belirli bir görevi icra etmek için iş birliği yapabildikleri gibi birbirinden bağımsız olarak da çalışabilirler. Bulut bilgi servisi (CIS) gelen görevleri uygun bir şekilde VM' lere dağıtan broker servislere sahiptir [11]. Bu tip servisler müşteri tarafından da geliştirilebilir ve bulut üzerinde ayrı bir VM üzerinde çalıştırılabilir. Her iki durumda da kullanılan görev çizelgeleme (Task Scheduling) algoritması performansı belirleyen en önemli faktördür.

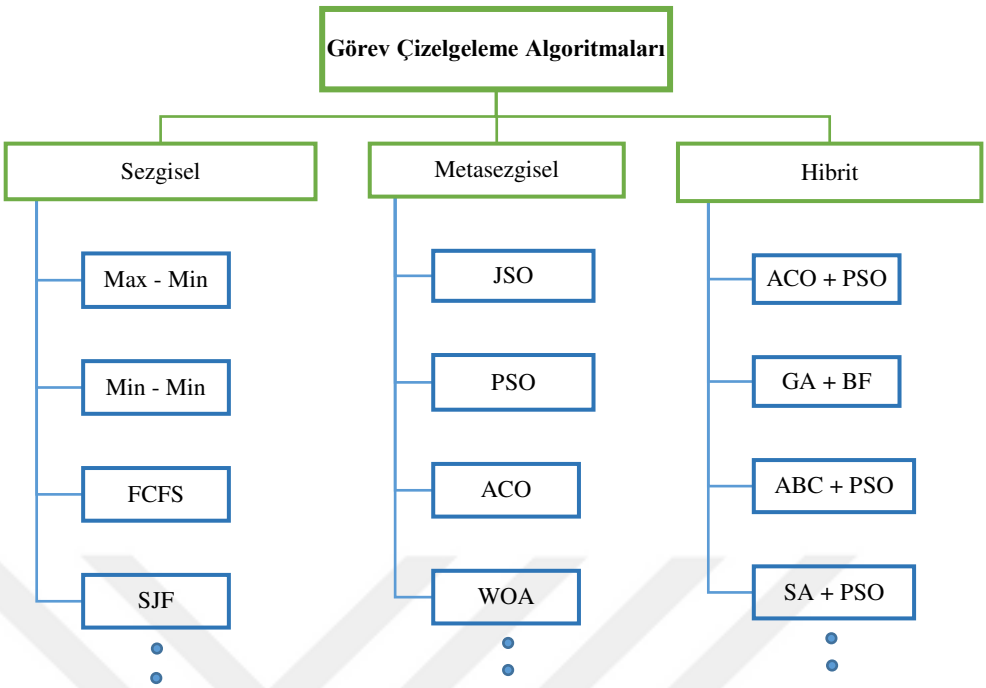
Görev planlanma, kaynak bulma, kaynak belirleme ve görev tahsisi olmak üzere üç ana mekanizmaya sahiptir [31]. Kaynakların bulunabilir olması paylaşılabilir kaynakların sorgulanabilir olmasını gerektirir. Kaynak belirlemesi, yapılan kaynak sorgulaması sonunda görevlerin özelliklerine göre en optimum kaynakların seçimini yapar. Görev tahsisi ise belirlenmiş kaynaklara ilgili görevleri gönderir ve takibini gerçekleştirir. Bulut sistem hem kaynak hem de müşteri çeşitliği açısından heterojen bir yapıya sahiptir. Bu da doğal olarak görev çizelgeleme optimizasyonunda farklı amaçların ortaya çıkmasına yol açmaktadır. Tamamlanma süresi (makespan), enerji tüketimi ve maliyet bu amaçlar içinde öne çıkmaktadır. Bu çeşitlilik dikkate alındığında görev çizelgelemede kullanılacak olan strateji doğru seçilmelidir. Literatürde önerilen görev çizelgeleme stratejileri Şekil 2.8' de gösterildiği gibi sınıflandırılabilir [32].



Şekil 2.8. Görev Çizelgeleme Stratejileri

Amaca göre görev çizelgeleme stratejisinde, tamamlanma süresi, maliyet, enerji tüketimi gibi hedeflerden biri veya birbiri ile çelişen birden fazlası dikkate alınabilir. Bulut altyapısı ve gelen görev tipi kullanılacak stratejinin statik mi yoksa dinamik mi olacağını da belirlemektedir. İş yükü değişiminin çok olmadığı sabit kaynaklı bulut sistemlerde statik bir çizelgeleme yapılabilir ancak günümüz bulut teknolojilerinin değişken yapıya sahip olmasından dolayı genellikle dinamik stratejiler tercih edilmektedir. Bu durum CIS mimarisinin heterojen yapısı için de geçerlidir. Planlama açısından sezgisel teknikler genellikle statik çizelgeleme için kullanılır. Sezgisel ve rastgele arama mekanizmasını birlikte kullanan metasezgisel algoritmalar dinamik sistemler için etkin çözümler üretmeyi başarabilir [9, 10, 13]. Bu çalışma da önerilen metasezgisel çözüm, heterojen ve dinamik bir CIS sistemde tek amaca yönelik geliştirilmiştir.

Görev çizelgelemesine uygun çözüm algoritması, sezgisel, metasezgisel veya hibrit algoritma türlerinden biri olarak belirlenir. Sezgisel algoritmalar statik çizelgeleme algoritmaları için kullanılır ve hızlı algoritmalardır. Ama büyük ölçekli dinamik ortama sahip bulut ortamları için yetersizdir. Metasezgisel algoritmalar çok zor problem türlerine dayalı olduğu için optimizasyon türlerini çözmek için en etkili yöntemler arasında yer almaktadır. Metasezgisel algoritmalar 'Sezgisel + Rastgele' şeklinde ifade edilebilir. Hibrit çizelgeleme algoritmaları, problemleri çözmek için iki algoritmanın birleşimi tekniğini kullanır ve bu da süre açısından dezavantaj sağlar. Şekil 2.9' da örnek görev çizelgeleme algoritmaları gösterilmiştir.



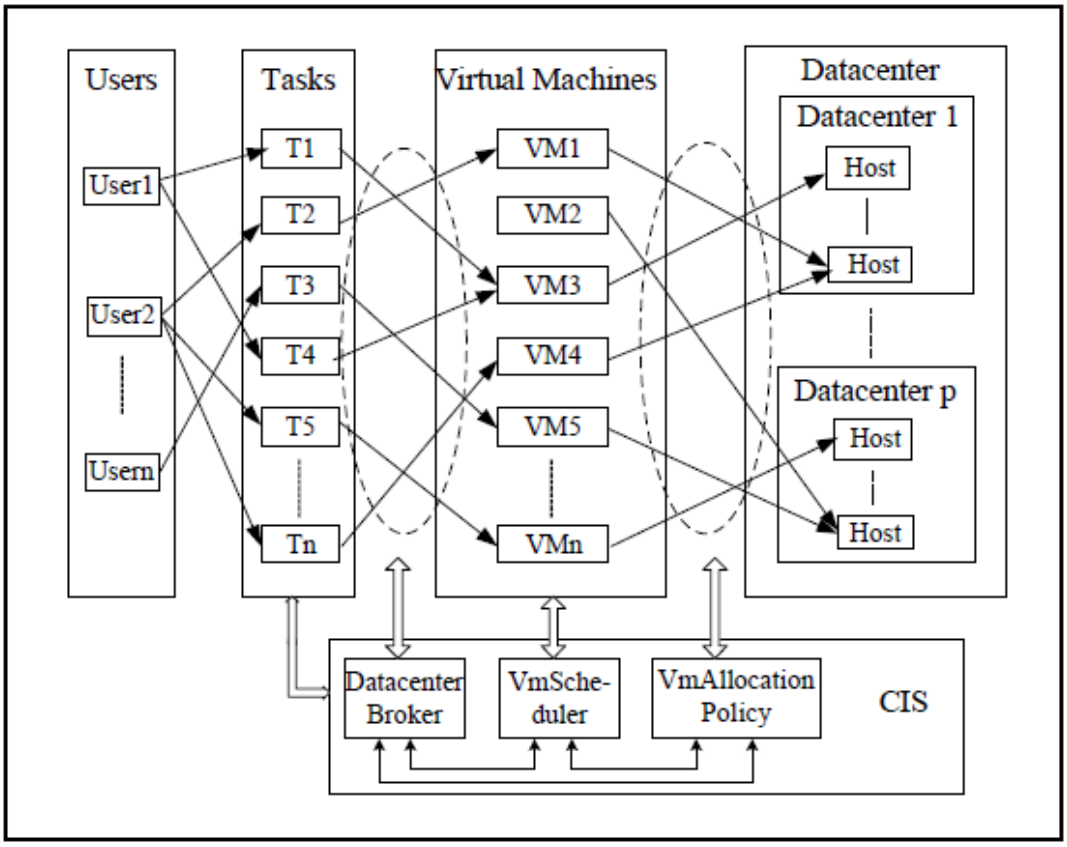
Şekil 2.9. Görev Çizelgeleme Algoritmaları

2.2.5. Simülasyon Ortamı ve CloudSim

Bulut bilişimde kaynak kullanımını iyileştirmek, yük dengelemek, enerji tüketimini azaltmak ve maliyetleri düşürmek araştırmacıların hedefleri arasındadır. Açık kaynak kodlu bulut benzetim ortamlarının kullanımı kodları inceleme, yeni algoritmalar geliştirme ve gerektiğinde iyileştirme yapmaya imkân tanıyabilmektedir. Pahalı fiziksel altyapı deneyleri yerine simülasyon ortamlarını kullanmak önemli avantajlar sağlamaktadır. Farklı alternatifleri olmakla beraber Cloudsim, bu amaca yönelik geliştirilmiş en bilinen açık kaynak kodlu bulut benzetim ortamıdır.

Bu tez çalışması da deneyler için Şekil 2.10' da çalışma prosedürü gösterilen CloudSim' i kullanmıştır. Cloudsim üzerinde görev dağıtım mekanizması bu şekilde daha iyi görülebilmektedir. Burada kullanıcıdan gelen m tane talebin n tane sanal makineye dağıtımı yapılmaktadır. Bulut Bilgi Servisi (CIS), kullanıcının isteklerini uygun bulut sağlayıcılarına eşler. CIS mimarisi içerisindeki 'DatacenterBroker', 'VmScheduler' ve 'VmAllocationPolicy' servisleri CloudSim' de çizelgelemenin temelini oluşturmaktadır [11].

a. **DatacenterBroker: K**ullanıcıların QoS ihtiyaçları baz alınarak servis sağlayıcılar ile arasındaki koordinasyonu sağlamaktan sorumlu olan Broker' ı modellemektedir. Broker, görevleri bulut üzerinde dağıtır. Kullanıcı tarafından geliştirilen çizelgeleme algoritmaları bu sınıfta kodlanır.



Şekil 2.10. CloudSim çalışma prosedürü [11]

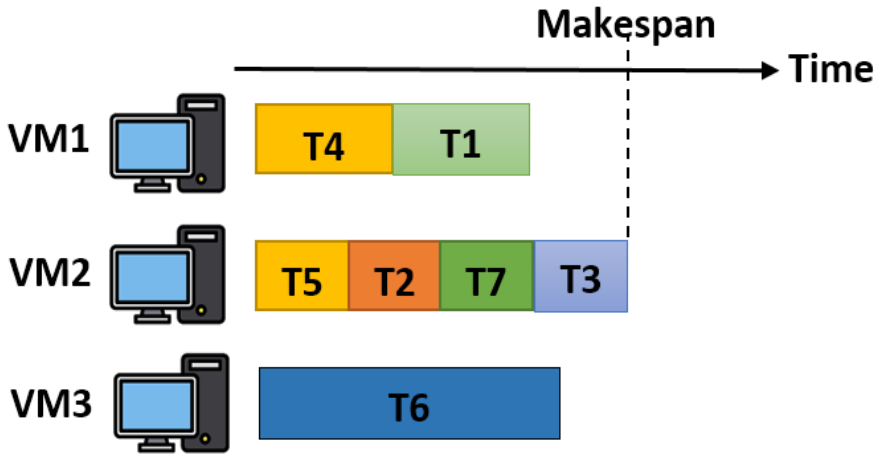
b. **VmScheduler:** Sanal makinelere gelen görevleri tahsis etmek için gereken politikaları temsil eder. Bu sınıfın fonksiyonelliği, belirli işlemci paylaşım politikalarına uyum sağlaması için kolaylıkla değiştirilebilir.

c. **VmAllocationPolicy:** Bir sanal makine gözlemcisinin sanal makineleri ana bilgisayarlara tahsis etmek için kullandığı tedarik politikasını temsil eder. Bu sınıfın temel işlevi, bir sanal makine yayılımı için bellek, depolama ve kullanılabilirlik gereksinimlerini karşılayan bir veri merkezindeki mevcut ve kullanılabilir ana bilgisayarı seçmektir.

3. MATERYAL VE METOT

Bu bölümde bulut sistemlerde görev çizelgeleme için önerilen yöntemin detayları ve kullanılan metasezgisel Denizanası Arama Algoritmasının (DAA/JSO) çalışma prensibi açıklanacaktır. Dolayısıyla ilk olarak JSO algoritmasının detayları verilecek ve yapılan ilk performans testlerinde kullanılan test (Benchmark) fonksiyonları tanıtılacaktır. Daha sonra tez kapsamında önerilen uyarlanmış JSO modeli ve diğer bir yaklaşım olan çoklu iş parçacığı ve çoklu süreç mekanizmalı JSO modeli tanıtılacaktır. Bu bölüm sadece kullanılan yöntem ve materyallerin detayını içermekte olup deneyler-sonuçlar bir sonraki bölümde ele alınacaktır.

Tez çalışmasının esas aldığı görev çizelgeleme, kuyrukta bekleyen görevleri (Task) Denizanası algoritmasının (JSO) çıktısına göre uygun sanal makinelere (VM) atanmasını sağlamaktadır. Kullanılan metasezgisel algoritma, bu görevlendirmeleri belirli amaç veya amaçlar doğrultusunda belirli hesaplamalarla gerçekleştirir. Ayrıca bu tez çalışması tek amaç (Single-objective) görev çizelgeleme optimizasyonuna odaklanmıştır ve amaç olarak tamamlanma süresini (makespan) dikkate almıştır. Tamamlanma süresi (makespan) kullanıcıdan gelen görevlerin sanal makinelere atanma gecikmesidir. Doğal olarak makespan optimizasyonunda amaç bu gecikmenin minimuma indirilmesidir. Optimizasyon amacının daha iyi anlaşılması için Şekil 3.1' de verilen örnek incelenebilir. Burada görev kuyruğunda bulunan 7 farklı görevin (Task) 3 farklı sanal makineye (VM) atanmış olduğu varsayılmıştır. Bu durumda makespan değeri, ikinci sanal makinenin (VM) görev tamamlanma süresine eşit olacaktır.



Şekil 3.1. Makespan Tanımı

Kullanılan JSO algoritması, makespan süresini farklı atama varyasyonları deneyerek minimuma düşürmeye çalışır. Bu varyasyonların oluşturulmasında, görev ve sanal makinelerin (VM) özelliklerine göre yapılan hesaplamalar dikkate alınır. Bu hesaplamalarda görevlerin komut

sayıları milyon komut (MI), sanal makinelerin (VM) hesaplama kabiliyetleri ise saniyede yürüttükleri milyon komut sayısı (MIPS) ile ifade edilir. $T_m = \{MI_0, MI_1, , . MI_K\}$ m.inci sanal makineye atanmış görevler olduğu kabul edilsin. Bu durumda k' nıncı görevin m' inci sanal makinede yürütüm süresi Denklem 3.1 ile hesaplanabilir. Tüm görevler için toplam yürütüm zamanı ise Denklem 3.2 ile bulunur.

$$ET_{km} = \frac{MI_k}{MIPS_m} \tag{3.1}$$

$$TET_m = \sum_{k=0}^{K} ET_{km} \tag{3.2}$$

Tüm sanal makineler (VM) dikkate alındığında, Denklem 3.3 ile yapılan hesaplamaya göre en yüksek toplam yürütüm zamanı aynı zamanda makespan değerini verecektir. Önerilen yöntemin uygunluk fonksiyonu (Fitness) Denklem 3.4' te ifade edildiği gibi minimum makespan değerini elde etmektedir.

$$Makepsan = max\{TET_m\}, \quad m \in \{VM_1, VM_2, \dots\} \tag{3.3}$$

$$F^{obj} = \min\{Makespan\} \tag{3.4}$$

3.1. Denizanası Arama Optimizasyonu (JSO)

Denizanası arama optimizasyonu; okyanustaki denizanalarının yiyecek bulmak gibi temel ihtiyaçlarını giderirken sergiledikleri hareketlerin, sürüdeki diğer bireyleri nasıl etkilediğini ve sürünün amacına daha kolay ulaşmak için ne tür davranışlar gösterdiğini anlamak için yapılan gözlemlerden esinlenerek geliştirilmiş bir optimizasyon algoritmasıdır [3].

3.1.1. Optimizasyonun Esin Kaynağı

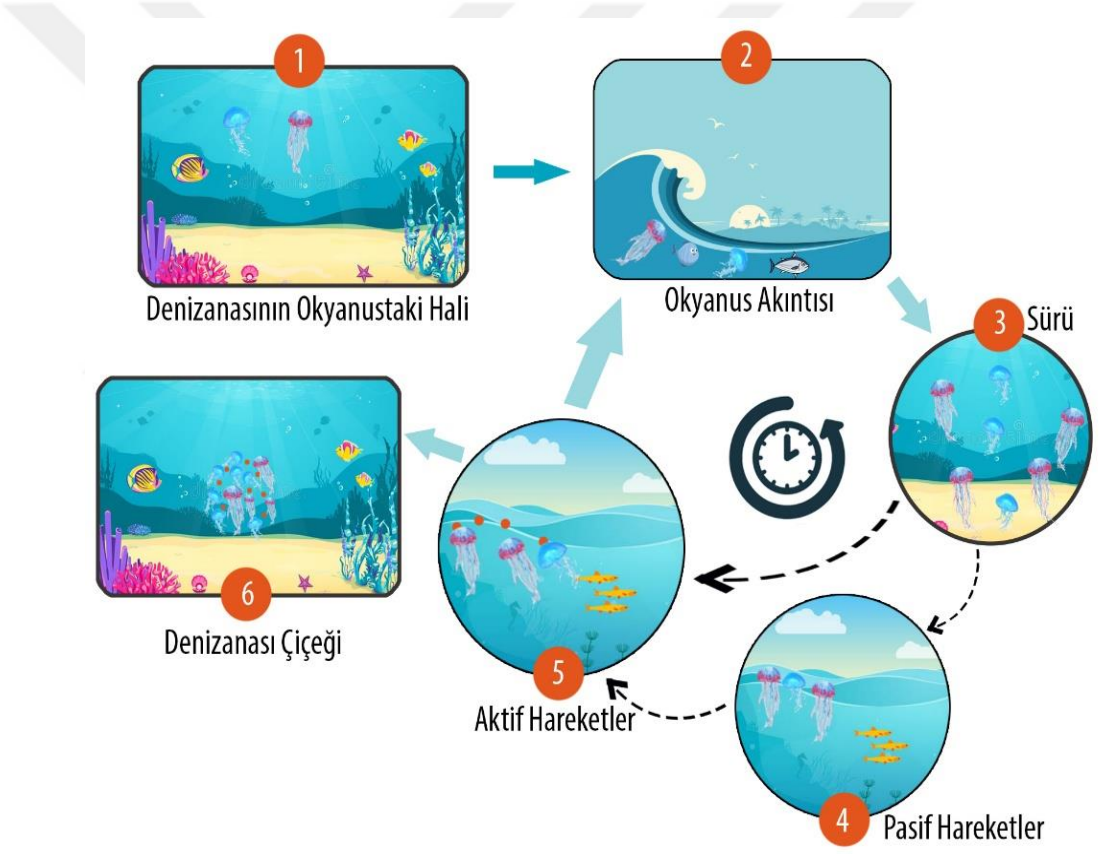
Denizanaları, çok çeşitli renk, boyut ve şekillere sahiptirler. Akıntı ve gelgitlere bağlı olarak çoğunlukla suda sürüklenirler. Okyanus akıntıları, mevcut besinler, oksijen mevcudiyeti, avlanma ve sıcaklık dâhil olmak üzere sürü oluşumunu çok sayıda faktör yönetir. Bu faktörler arasında, denizanasını bir sürü halinde toplayabildikleri için okyanus akıntıları oldukça önemlidir [3]. Denizanasının alt kısımları bir şemsiye gibi kapanıp açılır ve vücutlarını ileri iterek suda hareket ederler. Bu kabiliyetlerine rağmen çoğunlukla akıntıya bağlı olarak hareket ederler [33]. Koşulların uygunluğuna bağlı olarak denizanaları bir sürü oluşturabilir ve denizanası çiçeği olarak adlandırılan büyük denizanası sürüsü oluşturabilirler. Denizanaları zayıf organizmalar olduğundan karaya oturmamak için denizanası çiçeğini oluşturur [34].

Sürü oluşumunu okyanus akıntıları, mevcut besinler, oksijen, tahmin ve sıcaklık gibi çok sayıda faktör yönetir. Bu faktörler arasında sürüyü bir araya toplayabildikleri için okyanus akıntıları

en önemlisidir [34]. Denizanası tarafından ziyaret edilen yerlerdeki yiyecek miktarı değişir ve tüm konumlar için gıda oranları karşılaştırıldığında en iyi yer tespiti yapılır.

3.1.2. JSO' nun Matematiksel Modeli

Denizanası arama optimizasyonu, en ideal üç kurala dayanmaktadır; Denizanası ya okyanus akıntısını takip eder ya da sürü içinde hareket eder ve bu hareketler arasındaki geçişi bir zaman kontrol mekanizması yönetir. Denizanası yiyecek aramak için okyanusta hareket eder ve yiyecek miktarının daha fazla olduğu yerlere yönelir. Bulunan yiyeceğin miktarı, konuma ve ilgili amaç işlevine göre belirlenir [3]. Şekil 3.2 'de denizanasının okyanustaki bu davranışları gösterilmektedir.



Şekil 3.2. Denizanasının Okyanustaki Davranışları

Denizanası, her halükârda yiyecek miktarının daha fazla olduğu yere ($en\ iyi\ F^{obj}$) hareket etmek isteyecektir. Denizanası tarafından ziyaret edilen yerlerdeki yiyecek miktarı değişkenlik gösterebilir. Bu durumda yemek açısından en elverişli lokasyon, gıda miktarlarının karşılaştırılması ile bulunur. Denizanaları iki tip hareket sergilerler. Bunlar, okyanus akıntısı ile hareket ve sürü içerisinde harekettir. Bu hareketler arasındaki geçişi ise bir zaman kontrol mekanizması yönetir.

Şekil 3.3' te denizanası algoritmasının akış diyagramı gösterilmektedir [3]. Okyanus akıntısı çok miktarda besin içermektedir ve bu nedenle denizanasını daima kendine doğru çeker. JSO' da okyanus akıntısı ($\overrightarrow{OC}$), tüm aday çözümlerin (denizanaları) ortalama konumlarına ve en iyi uygunluk (Fitness) değerine göre bulunur. Denklem 3.5, okyanus akıntısı ile hareketin matematiksel modelini göstermektedir. Burada n_{pop} toplam popülasyonu, X^* o ana kadarki en iyi fitness değerine sahip denizanasını göstermektedir. Bu denklemde e_c çekicilik faktörüdür ve r_1 rastgele sayısı ve β hyper-parametresinin çarpımından oluşur ($e_c = \beta \times r_1, \quad r_1 \in [0,1]$).

$$\overrightarrow{OC} = \frac{1}{n_{pop}} \sum \overrightarrow{OC_i} = \frac{1}{n_{pop}} \sum (X^* - e_c X_i) = X^* - \beta \times r_1 \times \frac{\sum X_i}{n_{pop}} \tag{3.5}$$

Okyanus akıntısına göre hareket edilmesi durumunda, denizanalarının bir sonraki konumları ($X_i(t+1)$) Denklem 3.6 ile bulunur. Burada, $X_i(t)$, i' inci denizanasının mevcut durumunu, r_2 ise uniform rastgele sayısını temsil etmektedir.

$$X_i(t+1) = X_i(t) + r_2 \times \overrightarrow{OC} \tag{3.6}$$

Sürü içinde, pasif (A-tipi) ve aktif (B-tipi) olmak üzere iki tip hareket sergilenir [35]. Başlangıçta, sürü oluşturulurken çoğu denizanası A tipi hareket yapar. Zamanla, giderek B tipi hareket daha fazla oluşur. A-tipinde, denizanası kendi konumu etrafında rastgele (random) bir hareket gerçekleştirir. Bu hareket tipine ait model Denklem 3.7' de verilmiştir. Burada, γ hareket katsayısını, $U_b \; ve \; L_b$, sırasıyla arama uzayının üst-alt sınırlarını temsil etmektedir.

$$X_i(t+1) = X_i(t) + \gamma \times rand(0,1) \times (U_b - L_b) \tag{3.7}$$

B tipi hareket, i' inci denizanası ile sürü içerisinde rastgele seçilen j' inci denizanasının yiyecek kaynaklarına göre gerçekleşir. Hareket, yiyeceğin daha çok olduğu denizanasına doğru olacaktır. Denizanası belirlenen yöne doğru rastgele bir hareket gerçekleştirir. B tipinde, hareket yönü ve denizanasının yeni konumu Denklem 3.8 ve 3.9 ile hesaplanır. Burada, $\vec{D}$ hareket yönünü, r_3 ise uniform random değeri göstermektedir [3].

$$\vec{D} = \begin{cases} X_j(t) - X_i(t) & if \quad f(X_i) \geq f(X_j) \\ X_i(t) - X_j(t) & if \quad f(X_i < f(X_j) \end{cases} \tag{3.8}$$

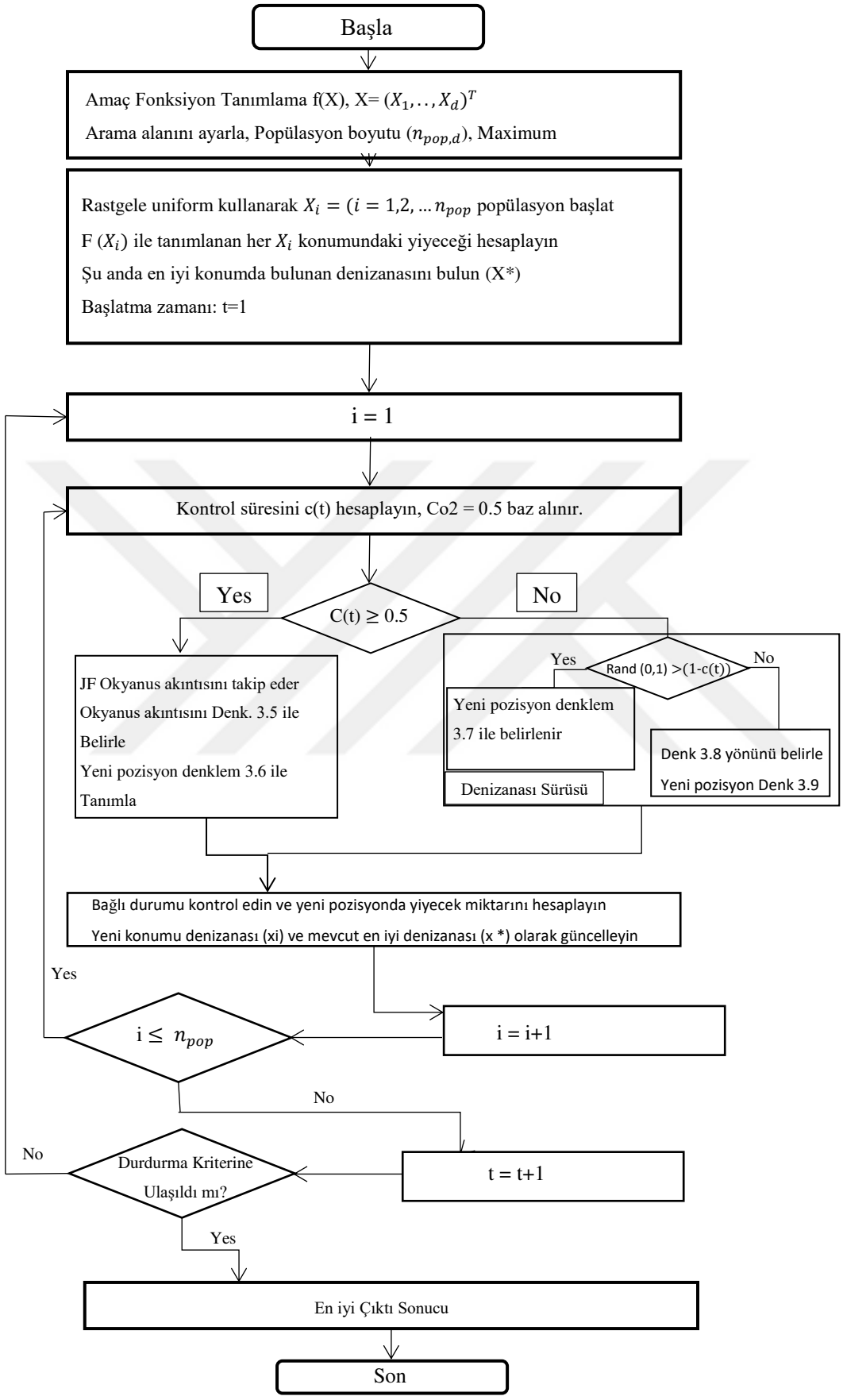
$$X_i(t+1) = X_i(t) + r_3 \, \vec{D} \tag{3.9}$$

Denizanasının sürü içerisindeki hareketleri, başlangıçta A tipi ile başlar ve zaman geçtikçe B tipine geçiş yapar. Bununla birlikte okyanus akıntısı hareketi de zaman içerisinde gerçekleşmektedir. JSO adaylarının tüm hareketleri için bir zaman kontrol mekanizması kullanılır. Bu kontrol mekanizması Denklem 3.10 ile modellenir. Bu modelde, t iterasyon sayısını, r_4 uniform

random katsayıyı ve $max_{iteration}$ maksimum iterasyon sayısını temsil etmektedir. Kontrol mekanizması, okyanustaki ortalama karbondioksit miktarını bir eşik değer (genellikle 0.5) olarak kabul eder ve karşılaştırma yapar. Kontrol fonksiyonunun değeri eşik değerin altında ise sürü içinde hareket gerçekleşir, eğer eşik değerin üzerinde ise okyanus akıntısına göre hareket gerçekleşir. Sürü içi harekette ise $1 - c(t)$ değeri dikkate alınır. Bunun için bir tekdüze (uniform) üreteç ile rastgele değer üretilir ve bu değer $1 - c(t)$ ile kıyaslanır. Üretilen rastgele değer daha yüksek ise A tipi, değilse B tipi hareket gerçekleşir.

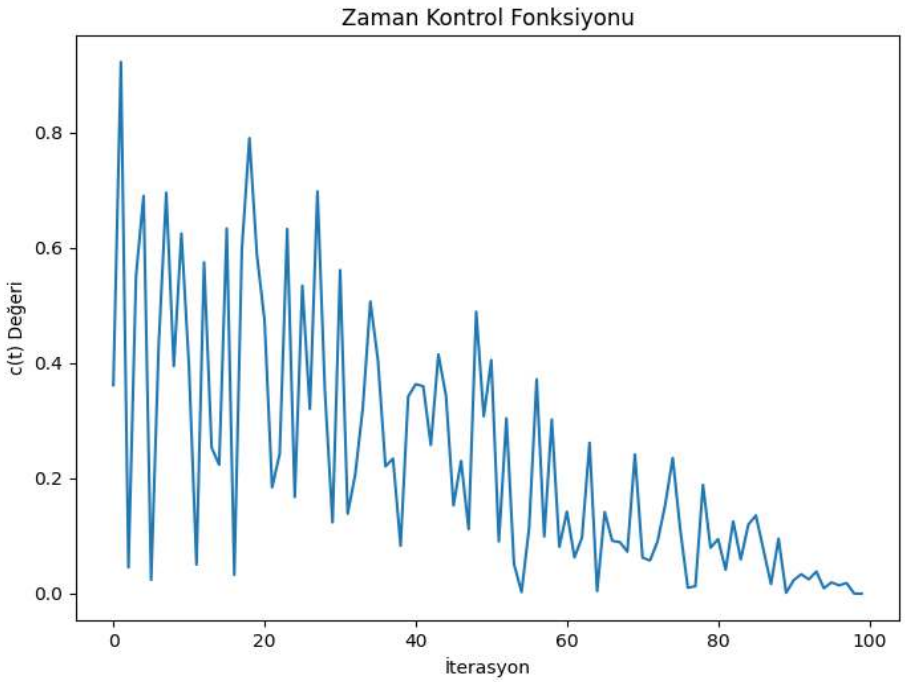
$$c(t) = \left|\left(1 - \frac{t}{max_{iteration}}\right) \times (2 \times r_4 - 1)\right| \tag{3.10}$$

JSO' da kullanılan katsayıların belirlenmesi için yapılan uygunluk testlerinde okyanus akıntısının etkisi ve hareket tiplerinin sonuçlara etkileri ayrıca gözlemlenmiştir. Bu testler sonucunda en optimum çözümlerin β = 3 ve γ = 0.1 için elde edildiği görülmüştür [3].



Şekil 3.3. Denizanası Algoritmasının Akış Şeması [3]

Zaman kontrol mekanizması, bir zaman kontrol fonksiyonu c(t) ve bir CO_2 sabiti içerir. Zaman kontrol işlevi, 0 ve 1 arasında sürekli değişken rastgele değerlere sahiptir. Şekil 3.4' te zaman kontrol mekanizmasının simülasyon çıktısı ve Tablo 3.1' de sözde kodu gösterilmektedir.



Şekil 3.4. Zaman Kontrol Mekanizmasının Simülasyonu

Tablo 3.1. Zaman Kontrol Mekanizmasının Sözde Kodu

Başla

for i in range (0, npop):

If (c(t) < 0.5) Zaman Kontrol Fonksiyonu Hesapla (**Denk**. 3.10)

($\overrightarrow{OC}$) okyanus akıntısı takip et (**Denk**. 3.5)

Else:

If 1- c(t) < rand (0,1)

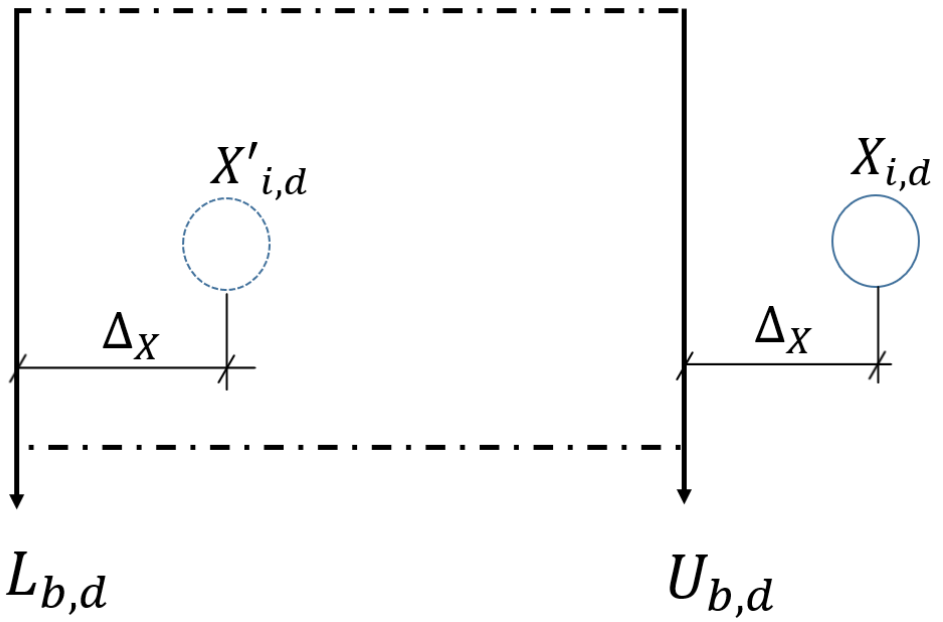
Denizanası A tipi hareket sergiler: (Pasif hareketler)

Else:

Denizanası B tipi hareket sergiler: (Aktif hareketler)

Bitir

Dünyada okyanusların dışında yaşam alanları olduğu için bir denizanası sınırlı arama alanının dışına çıktığında, adaylar son sınıra geri dönecektir. Şekil 3.5 ve Denklem 3.11 bu yeniden giriş sürecini göstermektedir.



Şekil 3.5. Yeniden Girme Yöntemi

$$\begin{cases} X'_{i,d} = (X_{i,d} - U_{b,d} + L_{b,d} & if \quad X_{i,d} > U_{b,d} \\ X'_{i,d} = (X_{i,d} - L_{b,d} + U_{b,d} & if \quad X_{i,d} < L_{b,d} \end{cases} \tag{3.11}$$

I' inci denizanasının d' ninci boyuttaki konumu; $X_{i,d}$ sınır kısıtlamaları kontrol edildikten sonra güncellenen konumdur. $U_{b,d}$ ve $L_{b,d}$ Sırasıyla arama uzaylarında d boyutunun üst sınırı ve alt kısmıdır.

3.2. Klasik JSO' nun İlk Performans Testleri ve Kullanılan Test (Benchmark) Fonksiyonları

Tez çalışmasında önerilen yöntemle görev çizelgeleme deneylerine geçmeden önce klasik JSO' nun performansı test edilmiştir. Literatürde, metasezgisel algoritmaların performansını değerlendirmek için kıyaslama fonksiyonları kullanılmaktadır [39]. Tercih edilen klasik JSO performansını görmek için farklı test (Benchmark) fonksiyonlar ele alınmıştır. Bu testler sonucunda başarısını ispatlayan Denizanası algoritması görev çizelgeleme problemleri için seçilmiştir. Kullanılan test fonksiyonları bu bölüm altında formüller ile birlikte açıklanmaktadır.

3.2.1. Sphere İşlevi

Sphere işlevi, sürüdeki her bir adayın karelerinin toplamını alınarak, tek boyutlu bir amaç fonksiyonu oluşturmaktadır. Global minimum aşamalı bir küre içerisindedir ve Denklem 3.14' te gösterilmektedir. Bu fonksiyonun matematiksel modellemesi Denklem 3.12' de ve Denklem 3.13' de arama sınırları gösterilmektedir.

$$f(x) = \sum_i^n x_i^2 \tag{3.12}$$

$$Arama\ Alanı = -\infty \leq x_i \leq \infty \tag{3.13}$$

$$Global\ minimum = f(x_1, \dots, x_n) = f(0, \dots, 0) = 0 \tag{3.14}$$

3.2.2. Rastrigin İşlevi

Rastrigin işlevi, matematiksel optimizasyon algoritmaları için performans testi problemi olarak kullanılan dışbükey olmayan bir işlevdir. Doğrusal olmayan çok modlu işlevin tipik bir örneğidir. İlk olarak 1974 yılında Rastrigin tarafından 2 boyutlu bir fonksiyon olarak önerilmiş ve Rudolph tarafından genelleştirilmiştir [49]. Bu fonksiyonun minimumunu bulmak, geniş arama uzayı ve çok sayıda lokal minimumu nedeniyle oldukça zor bir problemdir. Bu fonksiyonun matematiksel modellemesi Denklem 3.15' de ve Denklem 3.16' da arama sınırları gösterilmektedir. Ayrıca Denklem 3.17' de global minimumu gösterilmektedir.

$$f(x) = An + \sum_{i=1}^n [x_i^2 - Acos(2\pi x_i)] \tag{3.15}$$

$$Arama\ Alanı = -5.12 \leq x_i \leq 5.12\ \ A = 10 \tag{3.16}$$

$$Global\ minimum = f(0, \dots, 0) = 0 \tag{3.17}$$

3.2.3. Styblinski-Tang İşlevi

Styblinski-Tang işlevi, dalgalı arama uzayı ve çok sayıda lokal minimumu olduğu için oldukça zor bir problemdir. Bu fonksiyonun matematiksel modellemesi Denklem 3.18' de ve arama sınırları Denklem 3.19' da gösterilmektedir. Ayrıca Denklem 3.20' de global minimumu gösterilmektedir.

$$f(x) = \frac{\sum_{i=1}^n x_i^4 - 16x_i^2 + 5x_i}{2} \tag{3.18}$$

$$Arama\ Alanı = -5 \leq x_i \leq 5 \tag{3.19}$$

$$Global\ Minimum = X^* = (-2.903534, \dots, -2.903534) \tag{3.20}$$

3.2.4. Rosenbrock İşlevi

Rosenbrock işlevi, 1960 yılında Howard Rosenbrock tarafından tanıtılmıştır. Optimizasyon algoritmaları için performans test problemi olarak kullanılan dışbükey olmayan bir işlevdir [50]. Global minimum, uzun, dar, parabolik şekilli düz bir vadinin içindedir. Vadiyi bulmak önemsizdir ama global minimuma yaklaşmak zordur. Bu fonksiyonun matematiksel modellemesi Denklem 3.21' de ve Denklem 3.22' de arama sınırları gösterilmektedir. Ayrıca Denklem 3.23' te global minimumu gösterilmektedir.

$$f(x) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2] \tag{3.21}$$

$$Arama\ Alanı = -\infty \leq x_i \leq \infty \tag{3.22}$$

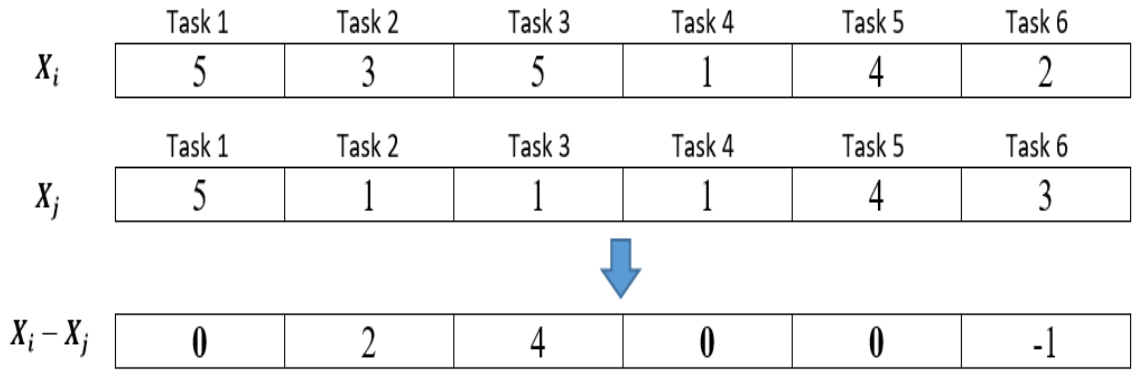
$$Global\ Minimum\ = X^* = (1, \dots, 1) = 0 \tag{3.23}$$

3.3. Klonlanabilir JSO (C-JSO) ve Görev Çizelgelemeye Uygulanması

Bu bölümde önerilen metasezgisel Klonlanabilir JSO Algoritması (K-JSO/C-JSO) ve onun bulut sistemlerde görev çizelgeleme problemlerine uygulanması açıklanmaktadır. Diğer metasezgisel algoritmalar gibi JSO' da görev çizelgeleme (Task Scheduling) problemlerinde lokal minimuma takılma riskine sahiptir. Bu riski aşmak için iterasyon veya popülasyon sayılarının artırılması, farklı rastgele üreteç fonksiyonlarının kullanılması gibi teknikler kullanılabilir. Yüksek popülasyon ve iterasyon sayıları beraberinde zaman maliyeti getirir. Bu çalışmada önerilen teknik, popülasyon artışının sezgisel ve dinamik olarak arttırılması ile daha kısa sürede optimum değere ulaşılabileceğini öne sürmektedir. Buna göre, belirli bir iterasyon boyunca en iyi değerde bir değişiklik olmuyorsa mevcut popülasyon çalışma esnasında belirli bir oranda arttırılır. Bu özellik aslında denizanalarının biyolojik özelliklerinden esinlenilmiştir. Doğada denizanaları kendilerini kontrollü bir şekilde klonlama yeteneklerine sahip canlılardır. Ancak burada kritik nokta yeni popülasyon adaylarının (klonların) konumsal değerleridir. Mevcut adaylara benzer adayların popülasyona eklenmesi lokal minimumdan çıkma olasılığını düşürecektir. Daha etkin bir keşif süreci için yeni bireylerin mevcutlardan farklı olması önerilen C-JSO algoritmasının temel noktalarından biridir. Yeni adayların benzerlik kontrolleri için kullanılacak fonksiyonun hesaplama maliyetinin yüksek olması özellikle yüksek popülasyonlarda algoritmanın zaman maliyetini de arttıracaktır. Bu nedenle hızlı ve etkin bir benzerlik kontrol fonksiyonun kullanılması gerekmektedir.

C-JSO, benzer aday üretimini önlemek için hızlı ve etkin bir benzerlik fonksiyonu kullanmaktadır. Bunu yaparken kullanılan görev çizelgeleme (Task Scheduling) problemi çözümünün ayrık yapıda olmasından faydalanmaktadır. Görev çizelgeleme algoritmasında görevler ve sanal makineler (VM) genellikle tam sayılar ile kodlanırlar. Şekil 3.6'da iki aday çözüm (X_i ve

X_j) için örnek bir kodlama gösterilmektedir. Aday nitelikleri bir görevi temsil etmektedir. Her bir nitelik değeri ise mevcut sanal makineleri (VM) gösteren tamsayı kodlarıdır. Görev çizelgeleme de benzerlik kriteri, mevcut ve yeni adayda aynı Task' lerin aynı VM' lere atanma sayısıdır. C-JSO, her görev için eşleşme karşılaştırması yaparak benzerlik oranı hesaplar. Bu hesaplamada benzerlik, iki adayın niteliklerinin farklarının alınması (aslında bu bir XOR işlemidir) sonucunda elde edilen toplam sıfır sayısının toplam görev sayısına oranı ile hesaplanır. Bu hesaplama Denklem 3.24 ve 3.25 ile ifade edilir. Buna göre Şekil 3.6'daki iki adayın benzerlik oranı 50%'dir.



Şekil 3.6. Görev Tahsisi İçin Örnek Bir Benzerlik Kontrolü

$$Df = X_i - X_j \tag{3.24}$$

$$Benzerlik = \frac{Df\ Sıfır\ Sayısı}{Görev\ Sayısı} \tag{3.25}$$

C-JSO' da popülasyon adayları (denizanaları) toplam görev sayısı boyutu kadar bir VM atama vektörüne sahiptir. Başlangıçta her aday için bu vektörlere VM' leri temsil eden rastgele tam sayı atamaları yapılır. Daha sonra her bir adayın uygunluk değeri Denklem 3.1- 3.4 ile hesaplanır. En iyi uygunluk değerine sahip aday, okyanus akıntısını da temsil edecektir. İterasyonlar boyunca her bir denizanası zaman kontrol fonksiyonun değerine bağlı olarak okyanus akıntısına göre sürü içinde yeni konumunu belirler. C-JSO' da en iyi değer değişimini izleyen bir binary değişken (φ) bulunmaktadır. Ön tanımlı sayıda (I) iterasyon sonunda en iyi değerde değişme olmazsa bu değişken doğru (True) değeri alır. Bu durumda mevcut popülasyon, ön tanımlı bir değer (θ) oranında arttırılır. Popülasyon arttırılırken yeni adayların benzerlikleri Denklem 3.24-3.25 ile hesaplanır. Hesaplama sonucunda, ön tanımlı benzerlik eşik değerinin (ε) altında olan adaylar popülasyona eklenerek temel adımlar tekrar edilir. Bu adımları gösteren C-JSO' a ait sözde kod (Pseudo-code) Tablo 3.2'de gösterilmektedir.

Tablo 3.2. C-JSO tabanlı görev çizelgeleme sözde kodu

Popülasyonu Başlat

Değişken Tanımla $\beta, \gamma, \theta, \varepsilon, I$ and $\varphi \rightarrow 0$

While (iterasyon < max iteration)

Denklem 3.1-4 tüm adayların uygunluk değerini hesapla

Denklem 3.10 zaman kontrol fonksiyonu hesapla

If c(t) < 0.5 okyanus akıntısı ($\overrightarrow{OC}$) takip et

Else if 1- c(t) < rand (0,1) A tipi hareket sergile

Else B tipi hareket sergile

If iterasyon değeri değişmezse ($\varphi \rightarrow 1$) **popülasyonu** $\%\theta$ oranında arttır Denk. 3.24-25

Else $\varphi \rightarrow 0$

End while

Return makespan

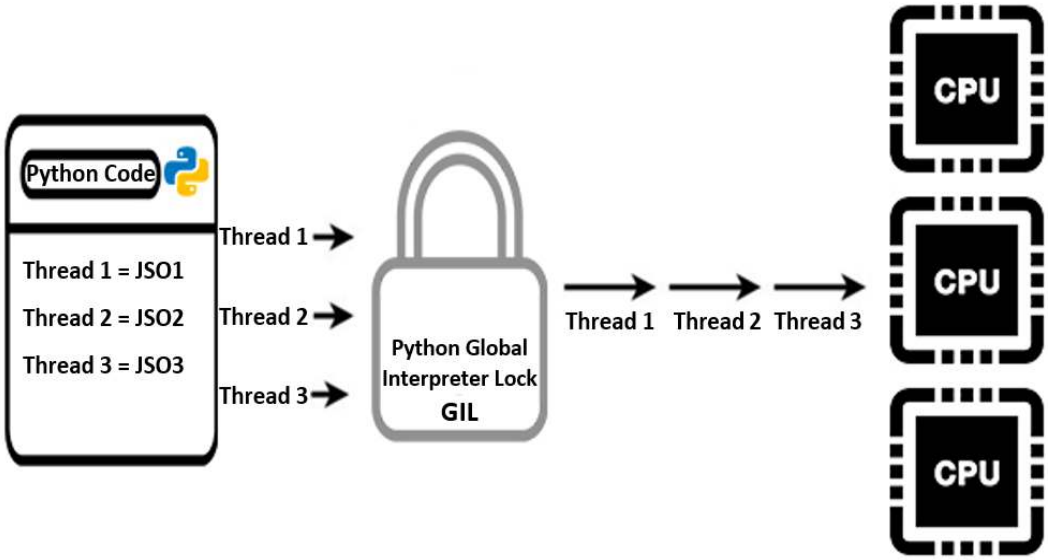
3.4. Görev Çizelgeleme Paralel Çalışma

Bu tez çalışması çoklu iş parçacığı ve çoklu süreç kullanan JSO yaklaşımını da görev çizelgeleme problemi için incelemiştir. Bunun için JSO algoritmasının çoklu iş parçacığı ve çoklu süreç versiyonları geliştirilmiştir. Daha sonra bu iki versiyon için JSO' nun performansı gözlemlenmiştir. Başarımı yüksek olan çoklu süreç temelli JSO için benzerliği düşük adaylar içeren popülasyon ile yeni bir yaklaşım önerilmiştir. Bu bölümde bu aşamalara ait detaylar sunulacaktır.

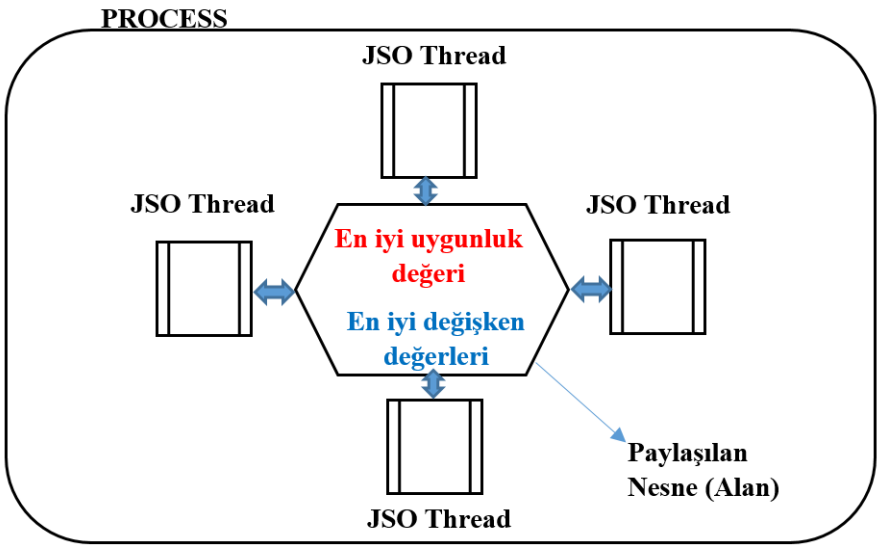
3.4.1. Çoklu İş Parçacığı (Multi-Thread) Temelli JSO

İş parçacıkları (Thread) bir Process' in birden fazla işi eş zamanlıya yakın sürede yapmasını sağlayan mekanizmalardır. Bir Process bünyesinde birden fazla Thread barındırabilir. Thread' ler aynı anda sadece tek bir iş yapabilir. Bir Process içerisinde birden fazla thread çalıştırılması Multi-Threading yürütümü gerektirir. Thread' ler çok çekirdekli işlemcilerde farklı çekirdeklerde eş zamanlı ve paralel olarak çalıştırılabilirler. Thread' ler yazılımsal olarak geliştirilebilirler ve bu nedenle pek çok programlama dili bu konuda destek sunmaktadır. Bununla birlikte programlama dillerinin sunmuş oldukları olanaklar pratikte farklı performans sonuçlarına yol açabilmektedir. Örnek olarak bu tez çalışmasında kodlama için tercih edilen Python dili, Thread yürütümleri için GIL (Global Interpreter Lock) isimli bir mekanizma içermektedir. GIL birden fazla Thread' in aynı çekirdek üzerinde eş zamanlı çalışmasına ve eş zamanlı kaynak paylaşımını kısıtlamaktadır [52]. Başka bir deyişle o andaki işlemci kaynakları sadece bir Thread' e verilmektedir. Şekil 3.7' de görüldüğü üzere Thread' ler GIL ile yürütüldüğünde bir dar boğaz oluşmaktadır. JSO' nun Multi-

Thread versiyonu için Python dilinin Thread Class' ı kullanılmış ve daha önce anlatılan JSO algoritması "run" fonksiyonunda yürütülmüştür. Multi-Thread JSO' a ait sözde kod, Tablo 3.3' te verilmiştir. Çoklu iş parçacığı mekanizması, literatürde de sıklıkla kullanılan en iyi değerin Thread' ler arasında iterasyonlar boyunca paylaşılması prensibine dayanmaktadır. Şekil 3.8' de gösterildiği gibi farklı Thread' ler üzerinde çalışan JSO' lar her bir iterasyonda paylaşılan nesneye (alana) kendi en iyi değerlerini yazmaya çalışır. Kendi değerlerini yazmadan önce paylaşılan alanda bulunan en iyi değerin kendi en iyi değerlerinden iyi olup olmadıklarını kontrol ederler. Eğer paylaşılan alandaki en iyi değer daha iyi ise ona ait değişken değerlerine sahip bir adayı kendi popülasyonuna katar. Aksi durumda paylaşılan alandaki değeri kendi en iyi uygunluk ve değişken değerleri ile günceller. Bu mekanizma aynı zamanda çoklu süreç (Multi-Process) için JSO' da uygulanmıştır. Tek fark paylaşılan alanın bir Process içinde değil hafızada ortak alanda olmasıdır. Doğal olarak kullanılan Process' ler arası iletişim mekanizması da farklıdır.



Şekil 3.7. Python Multi Thread çalışma prensibi



Şekil 3.8. Çoklu iş parçacığı (Multi-Thread) JSO çalışma prensibi

Tablo 3.3. Multi-thread JSO tabanlı görev çizelgeleme sözde kodu

Class MT- JSO> Thread:

Yapılandırıcı (Popülasyon, $\beta, \gamma, \theta, \varepsilon, I$ and $\varphi \rightarrow 0$, Kilit)**:**

run ()**:**

While (iterasyon < max iteration)

Denklem 3.1-4 tüm adayların uygunluk değerini hesapla

Denklem 3.10 zaman kontrol fonksiyonu hesapla

If c(t) < 0.5 okyanus akıntısı ($\overrightarrow{OC}$) takip et

Else if 1- c(t) < rand (0,1) A tipi hareket sergile

Else B tipi hareket sergile

Kilit mekanizmasını aktif et

Eğer en iyi uygunluk> paylaşılan alan en iyi: Paylaşılan alanı güncelle (Process içi)

Değilse: Popülasyondan en kötünün değerlerini paylaşılan en iyi değişkenlerle değiştir

Kilit mekanizmasını serbest bırak

End while

Return makespan

3.4.2. Çoklu Süreç (Multi-Process)

Süreçler (Process) işletim sistemi tarafından yönetilen yapılardır ve her bir Process kendisine ait bir hafıza alanı kullanmaktadır [51]. Yazılımsal olarak oluşturulabilirler ancak yönetimleri Multi-Thread' e göre karmaşıktır. Bunun ana sebebi farklı Process' ler farklı hafıza alanlarında çalıştıkları için Process' ler arası iletişim daha zordur. Ayrıca işletim sistemindeki içerik-anahtarlama (Context-Switching) daha karmaşıktır. Bununla beraber Python gibi dillerde (GIL mekanizması Process' ler için geçerli değildir) bazı durumlarda Multi-Thread' e göre daha performanslı olabilirler. Bu tez çalışmasında bu performans farkı da gözlemlenmek istendiğinden Multi-Process JSO versiyonu da yazılmıştır. Ana mekanizma Multi-Thread JSO' a benzemekle birlikte paylaşılan alan başka bir hafıza alanı olduğu için Python Process kütüphanesindeki Value ve Array veri yapıları kullanılmıştır (Şekil 3.9). Geri kalan sistem Multi-Thread mekanizması ile aynıdır. Multi-Process JSO' a ait sözde kod, Tablo 3.4' te verilmiştir.

Tablo 3.4. Multi-process JSO tabanlı görev çizelgeleme sözde kodu

Class MP_JSO-> Process:

Yapılandırıcı (Popülasyon**,** $\beta, \gamma, \theta, \varepsilon, I$ and, Kilit**):**

run **():**

While (iterasyon < max iteration)

Denklem 3.1-4 tüm adayların uygunluk değerini hesapla

Denklem 3.10 zaman kontrol fonksiyonu hesapla

If c(t) < 0.5 okyanus akıntısı ($\overrightarrow{OC}$) takip et

Else if 1- c(t) < rand (0,1) A tipi hareket sergile

Else B tipi hareket sergile

Kilit mekanizmasını aktif et

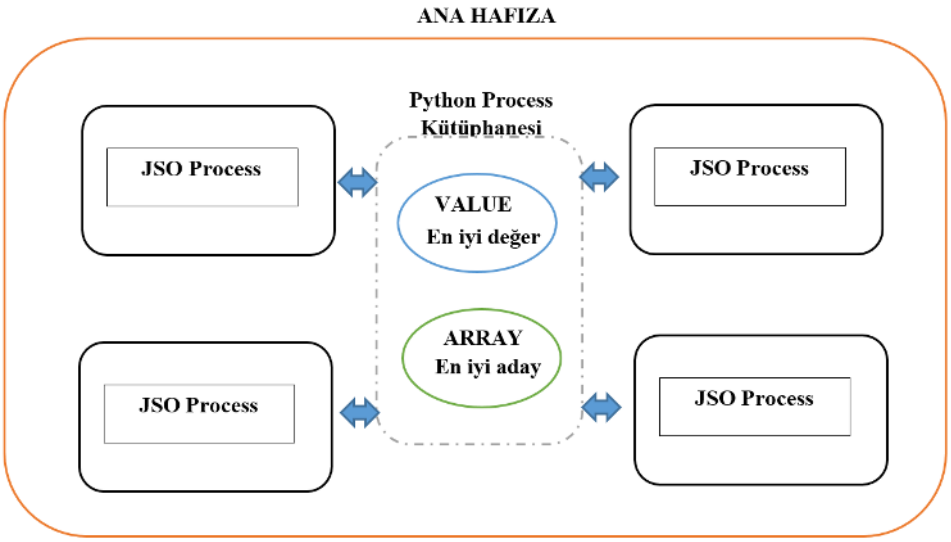
Eğer en iyi uygunluk> paylaşılan alandakinden iyi: Value ve Array güncelle

Değilse: Popülasyondan en kötünün değerlerini paylaşılan Array ile değiştir

Kilit mekanizmasını serbest bırak

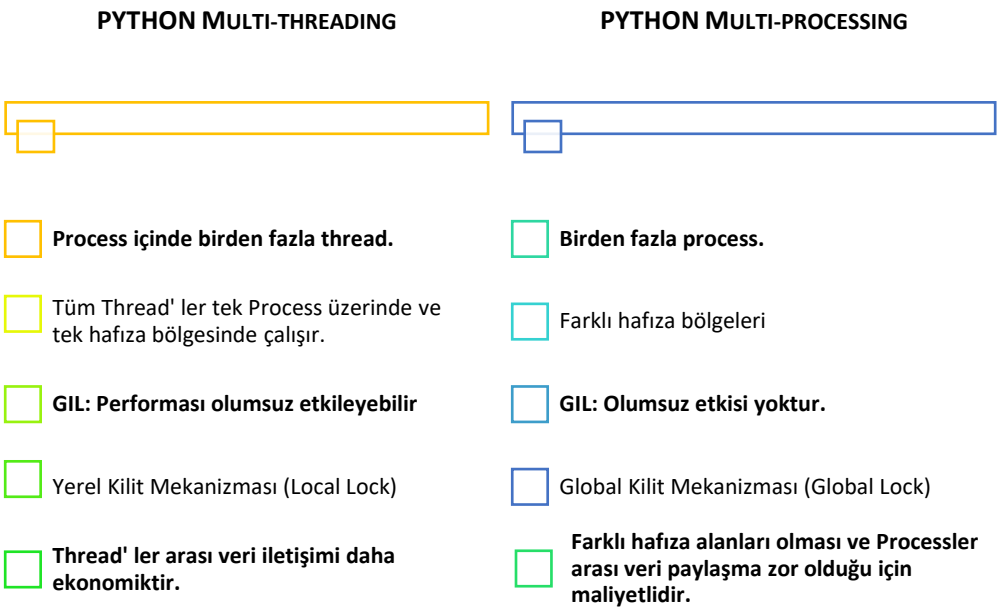
End while

Return makespan



Şekil 3.9. Çoklu süreç (Multi-Process) JSO çalışma prensibi

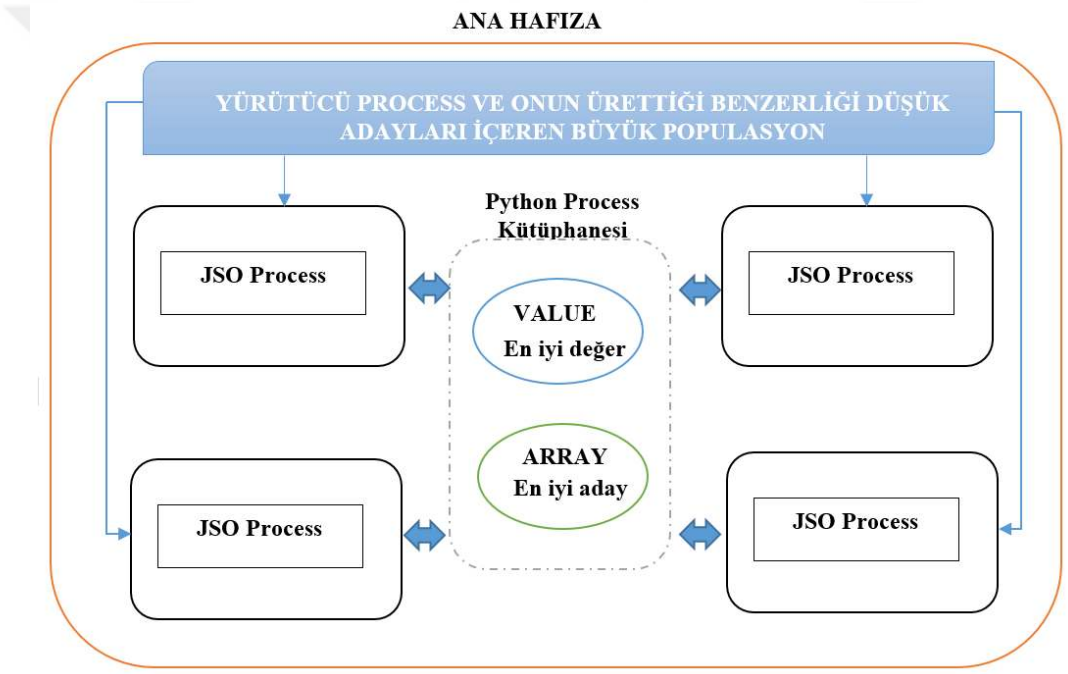
Karşılaştırma sonunda aday en iyi değeri paylaşılan alandaki değerden küçük ise 'VALUE' alanına atanır. Adayın minimum makespan değerinin bulunduğu konum 'ARRAY' alanına atanır. Eğer paylaşılan alan değeri daha küçük ise adaylar minimum değerine paylaşılan alandaki değeri atar. Ortak paylaşılan alana eş zamanlı müdahaleyi engellemek için Global kilit mekanizması kullanılmaktadır. Fikir vermesi açısından Şekil 3.10' da Multi-Processing ve Multi-Threading arasındaki farklar gösterilmiştir.



Şekil 3.10. Multi-Threading ve Multi-Processing Ayrıcalıkları

3.4.3. Benzerlik Kontrollü Multi Process

Daha önce çalışmalarımızda görüldüğü üzere görev çizelgeleme problemlerinde popülasyon sayısı arttığında yürütüm süresi olumsuz etkilenmektedir. Bu sorunu çözebilmek için JSO' nun paralel çalışma versiyonu kullanılmış ve sonuçları gözlemlenmiştir. Deneylerde görüleceği üzere Multi-Process JSO yani paralel versiyonu daha iyi performans göstermiştir. Bu nedenle farklı bir Multi-Process JSO algoritması geliştirilerek bu tez kapsamında önerilmiştir. Bu yeni yaklaşımda temel prensip benzerliği düşük adayların oluşturduğu büyük bir popülasyonun farklı Process' ler üzerinde çalışan JSO' lara dağıtılmasıdır. Benzersiz aday üretme mekanizması C-JSO' da ki gibi önerilen Denklem (3.24) ve (3.25)'i kullanmaktadır. Önerilen Multi-Process JSO yaklaşımı Şekil 3.11' de gösterilmiştir.



Şekil 3.11. Önerilen çoklu süreç (Multi-Process) JSO çalışma prensibi

4. BULGULAR VE TARTIŞMA

Deneyler üç aşamada gerçekleştirilmiştir. Birinci aşamada klasik JSO' nun performansını değerlendirmek için Benchmark fonksiyon testleri yapılmıştır. Test sonuçlarının başarılı olmasının ardından JSO ve C-JSO görev çizelgeleme problemlerine uygulanmıştır. C-JSO temelli görev çizelgeleme yönetiminin başarımı farklı bulut senaryoları için CloudSim simülatöründe [36] denenmiştir. C-JSO başarımı, varsayılan CloudSim görev çizelgeleme algoritması, klasik JSO ve Karınca Koloni Algoritması (ACO) algoritması sonuçları ile karşılaştırmalı olarak gösterilmiştir. JSO' nun paralel yaklaşımının deneyleri bu bölüm altında ayrı bir başlıkta karşılaştırmalı olarak verilmiştir.

4.1. Klasik Denizanası Algoritması (JSO) ve Benchmark Testleri

Klasik Denizanası Algoritmasının performansı (JSO), en iyi bilindik tek boyutlu Benchmark fonksiyonları ile test edilmiştir. Bu matematiksel işlev setleri, farklı senaryolar için karşılaştırılmıştır. Deneylerde her bir adayın farklı konumsal değerleri için değişken sayısı '5' olarak kullanılmıştır. Toplam popülasyon sayısı '50' ve tüm deneylerde iterasyon sayısı '100' olarak kullanılmıştır. Deney sonuçlarına bakıldığında en iyi adayın tüm test fonksiyonları için global minimuma yaklaşımında çok iyi başarım sağladığı görülmüştür.

Tablo 4.1. Benchmark fonksiyonları deney karşılaştırması

Karşılaştırma	1. İterasyon	25. İterasyon	50. İterasyon	75. İterasyon	100. İterasyon
Sphere Fonksiyon	5.840	1.805	0.900	0.172	0.172
Rastrigin Fonksiyon	9.033	3.031	2.035	1.22	1.228
Rosenbrock Fonksiyon	72.953	50.362	29.290	29.290	1.973
Styblinski-Tang Fonksiyon	57.987	0.772	0.772	0.772	-0.548

Klasik Denizanası Algoritmasının (JSO) performansı, Tablo 4.1 ve 4.2' de Benchmark fonksiyonları ile test edilmiş deney sonuçları gösterilmektedir. Yapılan testlerde tüm fonksiyonlar minimuma yaklaşma noktasında çok iyi başarım elde etmiştir. Bu deneyler sonucunda klasik

Denizanası Algoritması (JSO) başarısını ispatlamış ve görev çizelgeleme problemleri için tercih edilmiştir.

Tablo 4.2. Benchmark fonksiyonları farklı senaryolar için istatiksel sonuçlar

Karşılaştırma	**Medyan**	**Ortalama**	**Minimum**
Sphere Fonksiyon	0.279	0.415	0.172
Rastrigin Fonksiyon	1.495	2.456	0.141
Rosenbrock Fonksiyon	2.125	4.529	1.973
Styblinski-Tang Fonksiyon	-0.141	-0.350	-0.548

4.2. Görev Çizelgeleme ve Gerekli Simülasyon Parametreleri

Görev çizelgeleme deneyleri sonuçları simülasyon ortamında denenmiştir. Bu nedenle görev çizelgeleme deneyleri detaylarına geçmeden kullanılan simülasyon parametrelerinin verilmesi faydalı olacaktır. Simülasyonlar için öncelikle CloudSim' de bir veri merkezi oluşturulmuştur. Bu veri merkezinde her biri 16 GB ram, 10 TB Storage, 1 GB/s bant genişliği ve zaman paylaşımlı VM planlamasına sahip iki ana fiziksel sunucu bulunmaktadır. Tüm VM' ler bu fiziksel sunucular üzerine eşit paylaştırılmıştır.

Tablo 4.3. Simülasyon Parametreleri

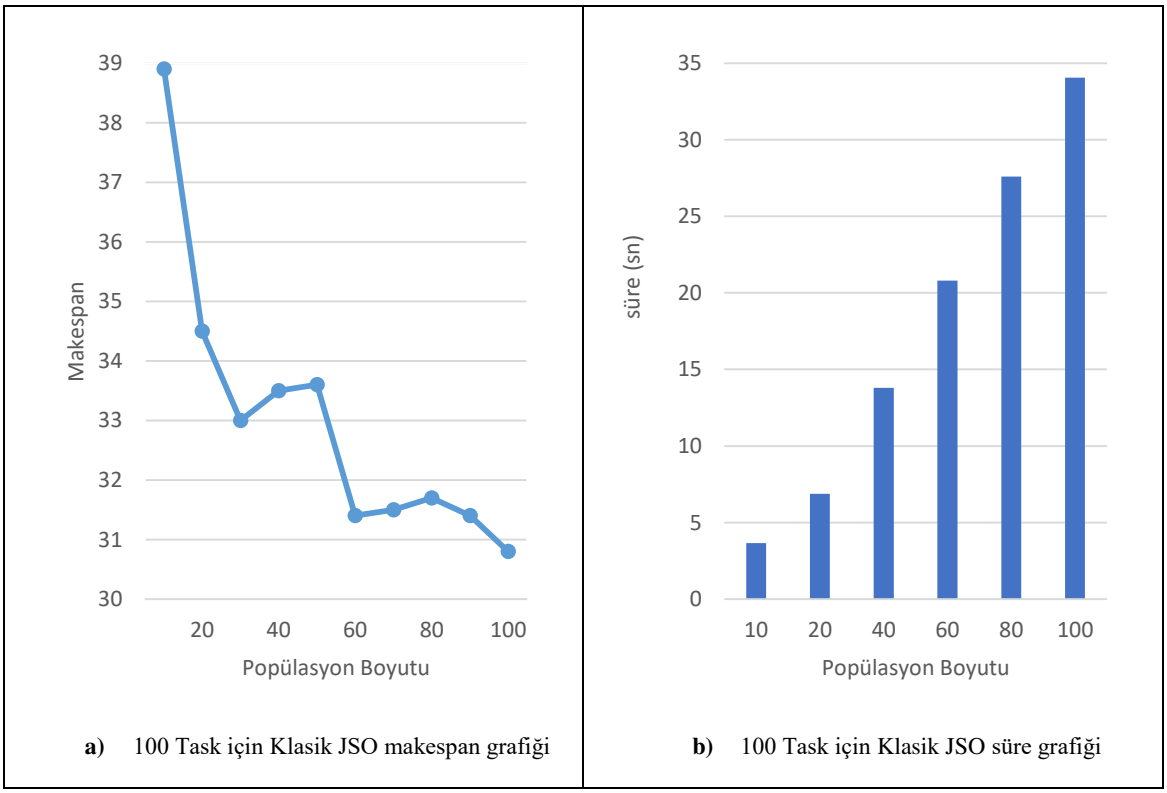
PARAMETRE	**DEĞER**
Popülasyon Boyutu	10, 20, 30, 50, 60, 80, 100
C-JSO Başlangıç Popülasyon Boyutu	10
Maximum Iterasyon	500
Task Sayısı	100 - 750
VM Sayısı	20
Görev Komut Uzunluğu (MI)	5000 - 20000
VM İşlem Kapasitesi (MIPS)	1000-5000
Popülasyon Artış Oranı C-JSO ($\boldsymbol{\theta}$)	13 %
Benzerlik Oranı (C-JSO) ($\boldsymbol{\varepsilon}$)	90 %

Bu bilgisayarlardan ilki 4 çekirdekli ikincisi ise çift çekirdekli X86 mimarisindeki işlemciye (CPU) sahiptir. Her işlemci çekirdeğinin işlem kapasitesi 10000 MIPS' tir. Bilgisayarlar üzerinde

Linux işletim sistemi ve Xen VMM vardır. Sanal makineler (VM) 512 MB ram, 10 GB Storage, 10 MB/s bant genişliği ve zaman paylaşımlı görev çizelgeleme konfigürasyonuna sahiptir. Sanal makinelerin (VM) işlem kapasitesi 1000 ile 5000 MIPS aralığında, görevlerin komut uzunluğu ise 5000 ile 20000 MI aralığında değişmektedir. Cloudsim' de standart görev çizelgeleme yöntemi "*CloudletSchedulerSpaceShared"*dir. Yapılan deneylerde kullanılan diğer parametreler Tablo 4.3' te verilmiştir. Farklı senaryoların performanslarını gözlemleyebilmek için 100, 250, 500, 750 görev sayıları ile gerçekleştirilen deneylerin her biri 10 defa çalıştırılarak istatistiksel sonuçlar elde edilmiştir.

4.3. Klasik Denizanası Optimizasyonu ile Görev Çizelgeleme (JSO)

Görev çizelgeleme deneylerinde ilk olarak popülasyon artışının makespan üzerine nasıl bir etki gösterdiği incelenmiştir. Bunun için, sabit boyutlu bir görev çizelgeleme probleminde farklı popülasyon sayılarına sahip klasik JSO temelli çözümler ve süreleri analiz edilmiştir. Şekil 4.1.a-b, yapılan bu deneylere ait sonuçları göstermektedir. Bu deneylerde, 100 görev için farklı popülasyon boyutlarına sahip simülasyonlar çalıştırılmıştır. Şekil 4.1.a'dan görüleceği üzere popülasyon artışı makespan değerinde bir iyileşmeye yardımcı olmaktadır. Ancak öte yandan Şekil 4.1.b bu iyileşmenin çözüm süresinde olumsuz etkiye sahip olduğunu göstermektedir.



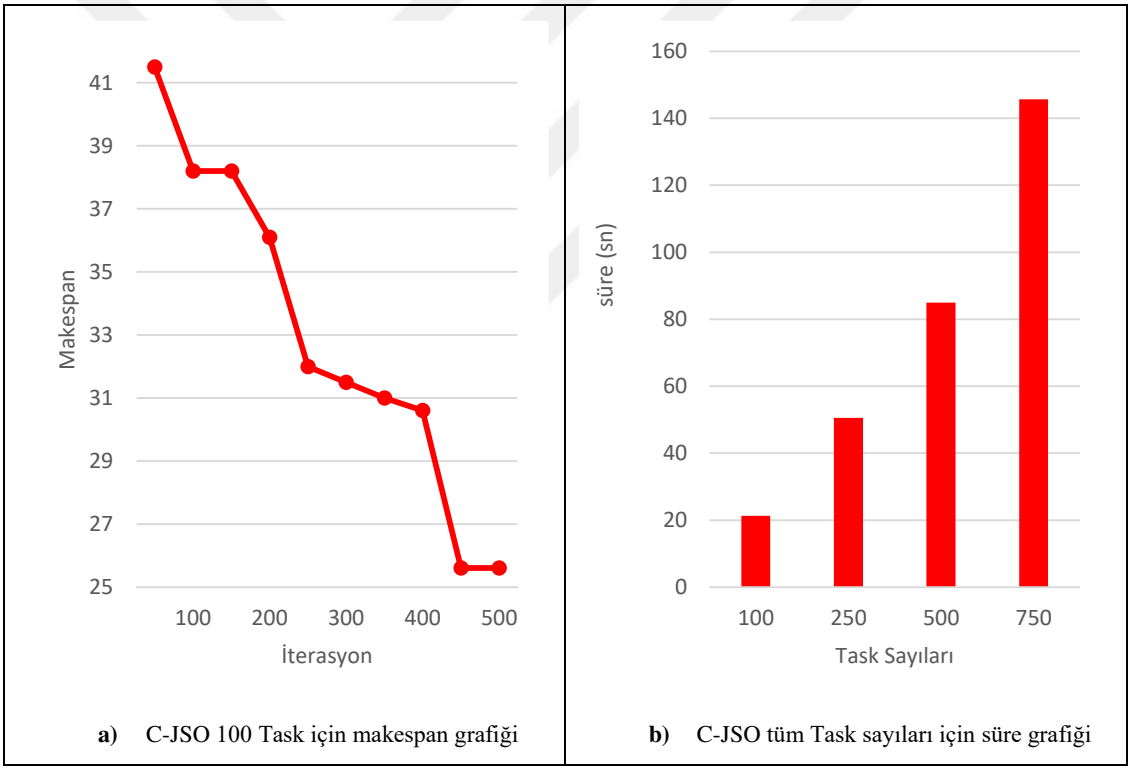
a) 100 Task için Klasik JSO makespan grafiği

b) 100 Task için Klasik JSO süre grafiği

Şekil 4.1. Klasik JSO farklı popülasyon boyutlarına göre makespan ve süre sonuçları

4.4. Klonlanabilir Denizanası Algoritması ile Görev Çizelgeleme (C-JSO)

Klasik JSO' da çözüm süresinin çok yüksek çıkmasından dolayı, bu çalışmada C-JSO algoritması alternatif olarak önerilmiştir. JSO ortaya çıkan lokal minimum problemlerini daha kısa sürede aşma noktasına odaklanmamıştır. C-JSO literatürdeki örneklerinden farklı olarak durum kontrollü dinamik popülasyon değişkenliği ile mevcut çözümlere daha hızlı ulaşmayı amaçlamıştır. Böylece algoritma, belirli görev boyutlarında, optimum sonuca uygun popülasyon sayısı ile daha kısa sürede ulaşılabilecektir. Şekil 4.2.a'dan görüleceği üzere C-JSO makespan iyileşmesi klasik JSO' ya göre daha başarılı olmuştur. Şekil 4.2.b C-JSO' nun tüm task sayıları için süre karşılaştırması verilmiştir. C-JSO ile kullanılan bu çizelgeleme problemi tamamlandığında, son popülasyon boyutu '100' olarak sonlanmıştır. Dolayısıyla C-JSO' da ve JSO' da popülasyon boyutu '100' olarak baz alındığında, C-JSO süre açısından da başarısını ispatlamıştır.

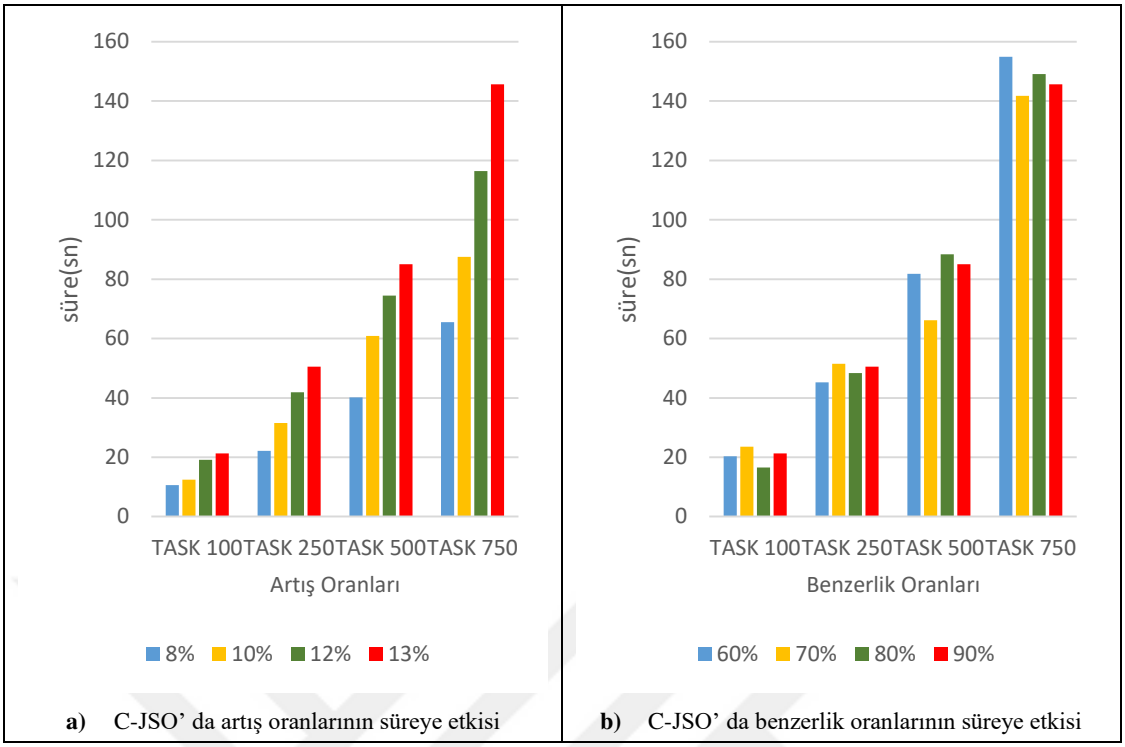


a) C-JSO 100 Task için makespan grafiği

b) C-JSO tüm Task sayıları için süre grafiği

Şekil 4.2. C-JSO makespan ve süre sonuçları

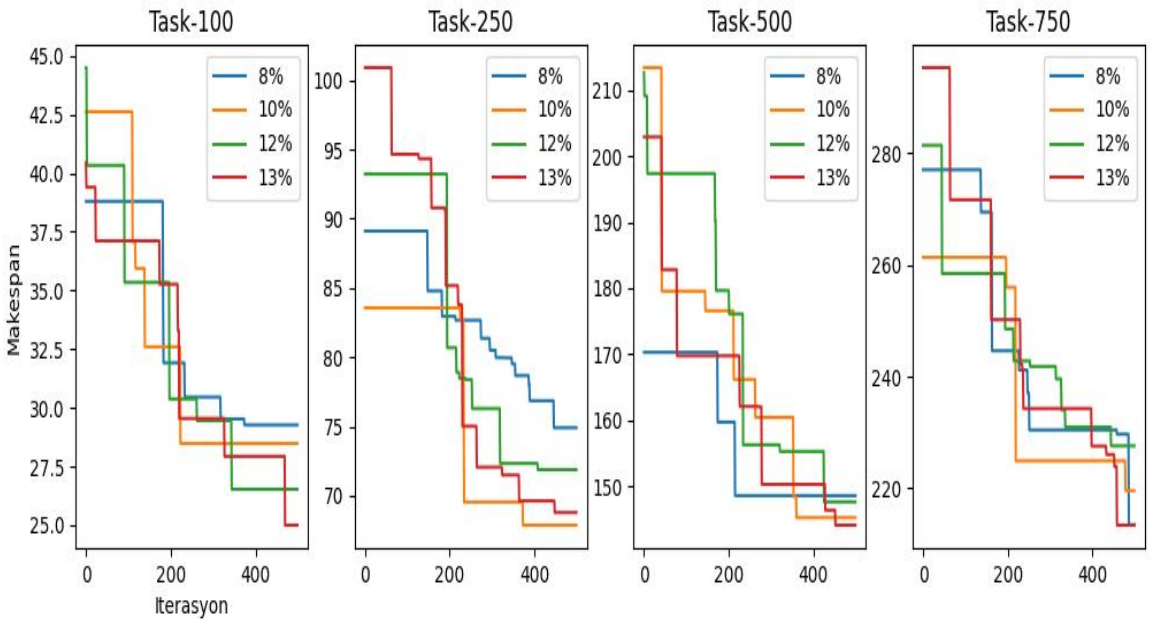
Klonlanabilir denizanası algoritmasının (C-JSO) önemli parametreleri, popülasyon artış oranı ve yeni aday üretiminde kullanılan benzerlik oranıdır. Bu sebeple, farklı değerler için parametre deneyleri gerçekleştirilmiş ve C-JSO' nun davranışı incelenmiştir. Deneylerde artış oranı (increase rate) için, %8, %10, %12 ve %13 için değerleri seçilmiştir. %13'ün üzerindeki deneylerde makespan değerlerinde iyileşme olmadığı gibi hesaplama süresi de JSO' a yaklaşmıştır. Benzerlik oranları içinde %60, %70, %80 ve %90 seçilmiştir.



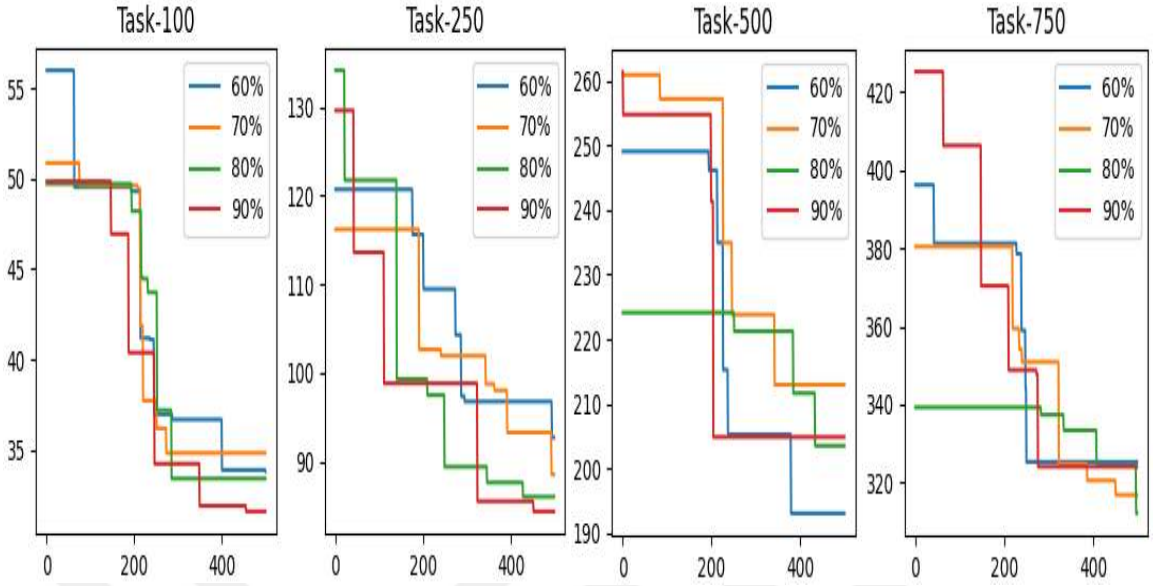
a) C-JSO' da artış oranlarının süreye etkisi **b)** C-JSO' da benzerlik oranlarının süreye etkisi

Şekil 4.3. Tüm görevler için artış ve benzerlik oranlarının süreye etkisi

Deneylerde en iyi sonuçlar, artış oranı (increase rate) %13 ve benzerlik oranı (similarity rate) %90 oranlarında elde edilmiştir. Bu parametrelerin farklı senaryolar için ortalama süre performansları Şekil 4.3.a-b'de verilmiştir. Artış ve benzerlik oranlarının makespan üzerine etkileri sırasıyla Şekil 4.4 ve 4.5'de verilmiştir.

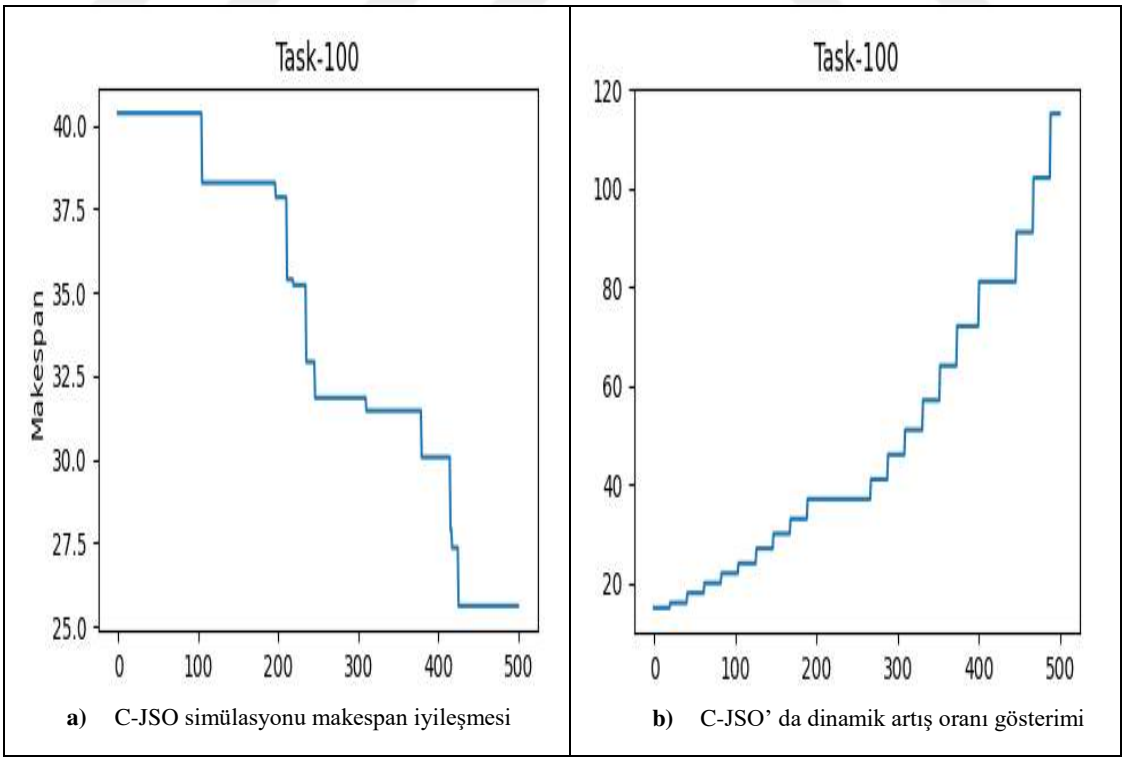


Şekil 4.4. Tüm görevler için artış oranlarının makespan etkisi



Şekil 4.5. Tüm görevler için benzerlik oranlarının makespan etkisi

Şekil 4.6.a-b'de C-JSO algoritmasının, makespan iyileştirmesi ve dinamik popülasyon artışı verilmiştir. Şekil 4.6.b' de gösterildiği gibi dinamik popülasyon artışı iterasyonlar boyunca devam etmektedir.



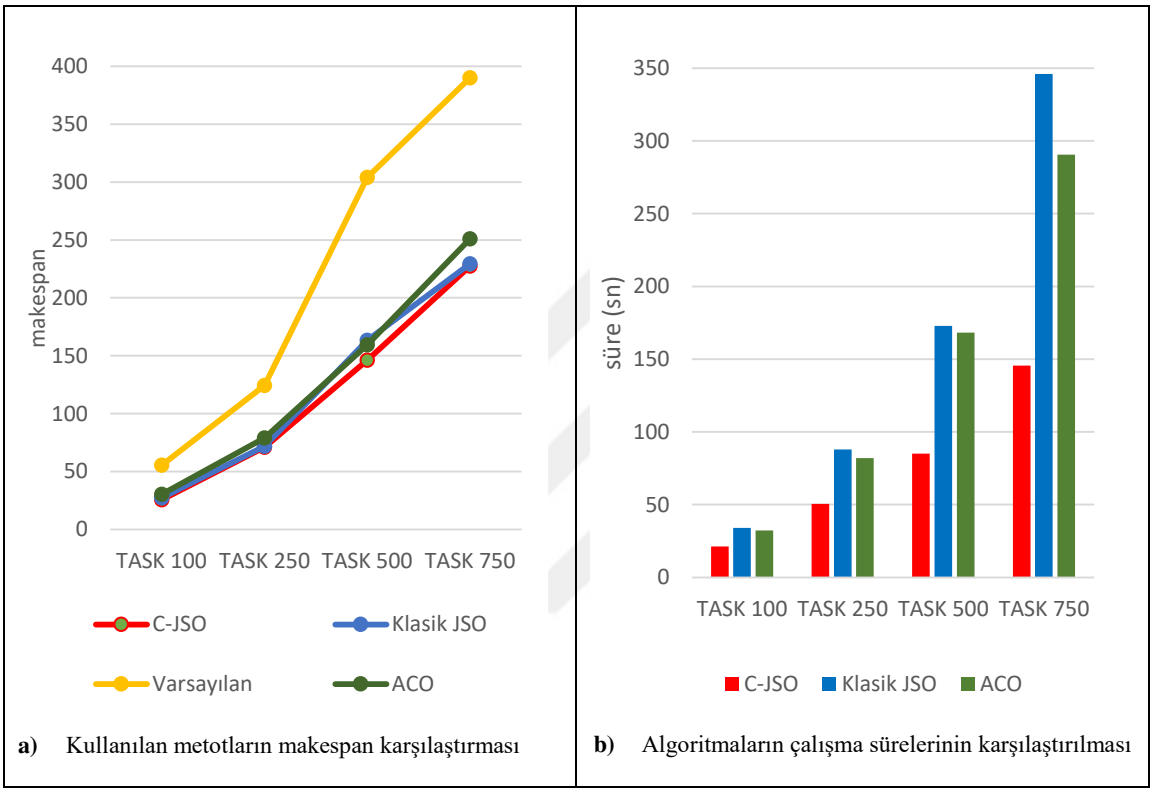
a) C-JSO simülasyonu makespan iyileşmesi

b) C-JSO' da dinamik artış oranı gösterimi

Şekil 4.6. C-JSO örnek simülasyon çıktıları

4.5. Görev Çizelgelemede Kullanılan Tüm Yöntemlerin Karşılaştırılması

Bu bölümdeki deneyler, CloudSim senaryoları için yapılmıştır. Bu deneylerde klasik JSO, C-JSO, varsayılan bulut çizelgeleme ve Karınca Koloni Optimizasyonu (ACO) temelli yöntemler çalıştırılmış ve sonuçları incelemiştir. Yapılan bu deneylere ait karşılaştırmalı ve istatistiksel sonuçlar sırasıyla Şekil 4.7.a-b' de ve Tablo-4.4'te verilmiştir.



a) Kullanılan metotların makespan karşılaştırması

b) Algoritmaların çalışma sürelerinin karşılaştırılması

Şekil 4.7. Kullanılan yöntemlerin makespan ve süre karşılaştırması

Simülasyon deneylerinde en kötü makespan değerleri varsayılan çizelgeleme algoritması tarafından elde edilmiştir. Metasezgisel yaklaşımlar, makespan değerlerinde birbirine yakın ve varsayılan çizelgeleme algoritmasına göre yaklaşık iki kat daha başarılı sonuçlar yakalamıştır. Kendi aralarında ise JSO ve C-JSO, ACO temelli çizelgelemeye göre nispeten daha başarılı olmuştur. Süre açısından incelendiğinde C-JSO, diğer algoritmalara göre çok daha başarılıdır. JSO hesaplama süresi açısından en yüksek değerlere sahip iken C-JSO ise en kısa süreyi elde etmeyi başarmıştır.

İstatiksel sonuçlara göre ACO algoritması her ne kadar minimum makespan değerlerinde JSO ve C-JSO değerlerine yaklaşmışsa da metasezgisel yöntemler içinde en yüksek makespan değerleri de yine ACO tarafından elde edilmiştir. İstatiksel sonuçlara göre JSO algoritması makespan bazında C-JSO' ile yakın değer vermesine rağmen hesaplama süresinde C-JSO' nun

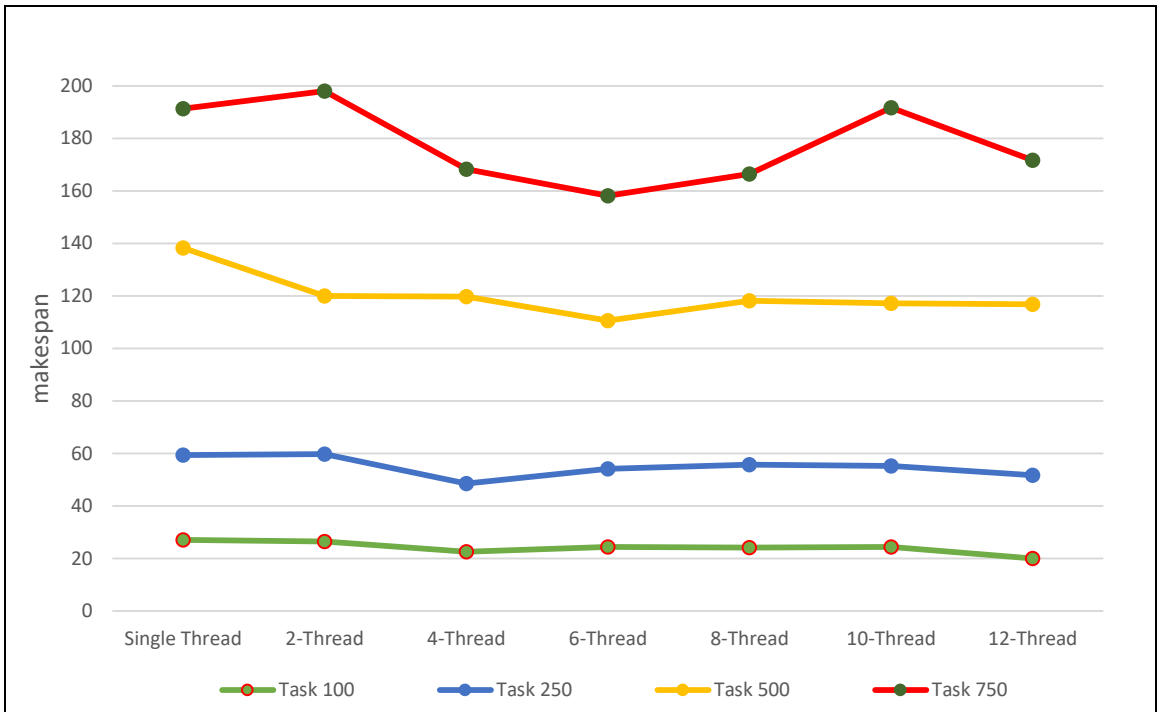
oldukça gerisinde kalmıştır. C-JSO' nun dinamik popülasyon artışı mekanizması sayesinde asıl üstünlüğü, hesaplama süresinde ortaya çıkmaktadır.

Tablo 4.4. Kullanılan metotların istatistiksel sonuçları (Makespan)

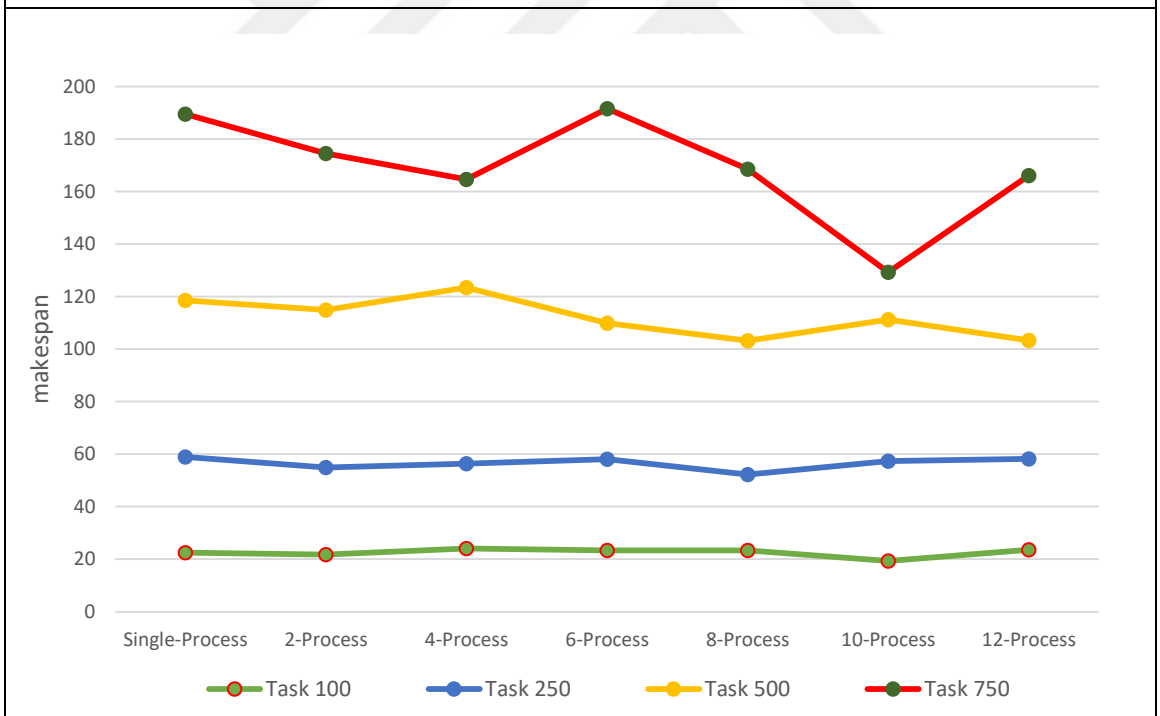
	Metot	100 Task	250 Task	500 Task	750 Task
Minimum	JSO	27.38	71.92	163.46	229.43
Maximum		31.05	77.92	201.73	257.34
Mean		29.33	73.81	179.72	244.08
Median		29.64	75.76	177.28	244.68
Std.		1.26	4.08	12.54	8.96
Minimum	C-JSO	25.61	71.06	146.63	227.45
Maximum		31.42	77.09	161.66	255.74
Mean		27.96	73.64	154.86	242.86
Median		27.88	74.07	154.72	242.46
Std.		1.48	1.79	4.03	6.94
Minimum	ACO	30.31	79.02	159.54	251.10
Maximum		35.16	87.33	193.47	287.11
Mean		34.53	83.18	171.11	267.01
Median		32.14	84.06	177.76	263.74
Std.		2.29	3.12	10.49	9.88
Minimum	CloudSim	55.51	124.43	304.13	390.08

4.6. Görev Çizelgelemede Paralel Yaklaşım Deneyleri

Bu bölümde gerçekleştirilen paralel JSO deneylerinin sonuçları tartışılmıştır. İlk olarak Multi-Thread ve Multi-Process JSO için genel performans karşılaştırmaları yapılmıştır. Karşılaştırmalar basitlik açısından sabit popülasyon boyutuna göre yapılmıştır. Bu ilk deneylerde asıl amaç, bulut sistemde görev çizelgeleme problemi için paralel JSO uygulamasında Python GIL' in etkisinin nasıl olduğunu görmektir. Şekil 4.9.a-b' de her bir Process ve her bir Thread için 10 popülasyon ataması yapılmıştır. Görüldüğü üzere süre bazında Multi-Process sonuçları neredeyse iki kat başarım sağlamıştır. Bunun da nedeni daha önceki bölümlerde açıklandığı gibi Python programlama dilinin GIL fonksiyonunun Multi-Thread' i olumsuz etkilemesiydi. Şekil 4.8.a-b' de Multi-Thread ve Multi-Process makespan sonuçları karşılaştırmalı olarak verilmiştir. Burada Multi-Process özellikle yüksek Task sayılarında daha iyi başarım sağlamıştır. Multi-Process sonuçları hem süre hem de makespan bazında daha iyi sonuçlar verdiği için önerilen paralel JSO için Multi-Process versiyon kullanılmıştır.

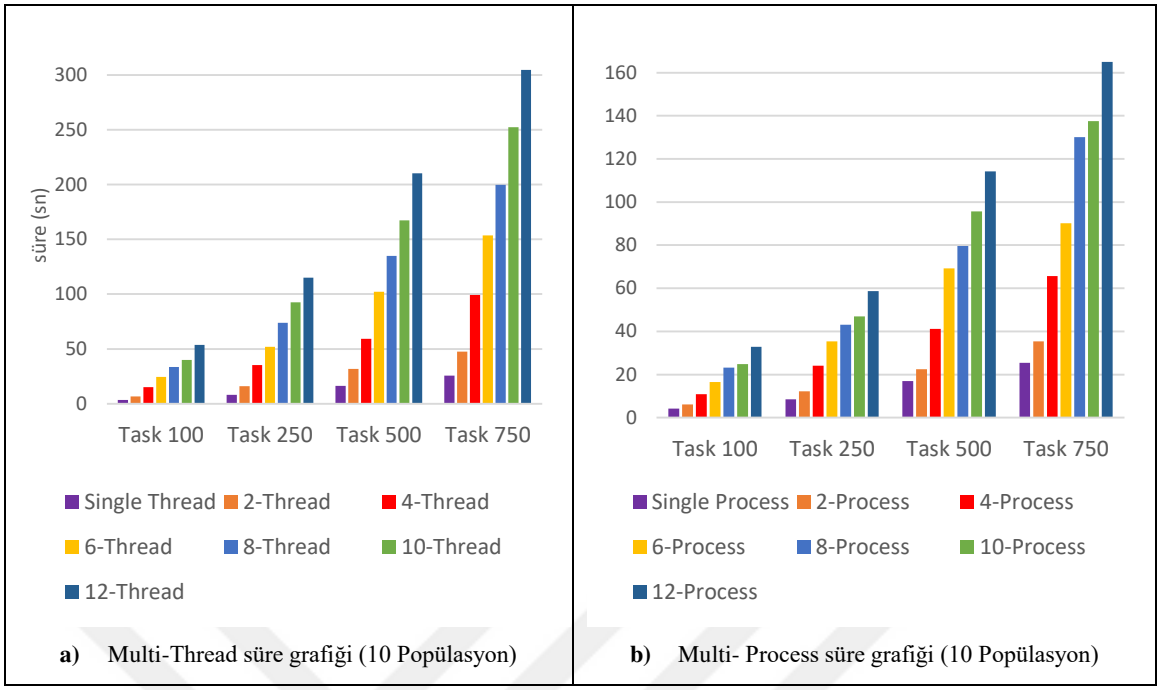


a) Multi-Thread makespan grafiği (10 Popülasyon)



b) Multi-Process makespan grafiği (10 Popülasyon)

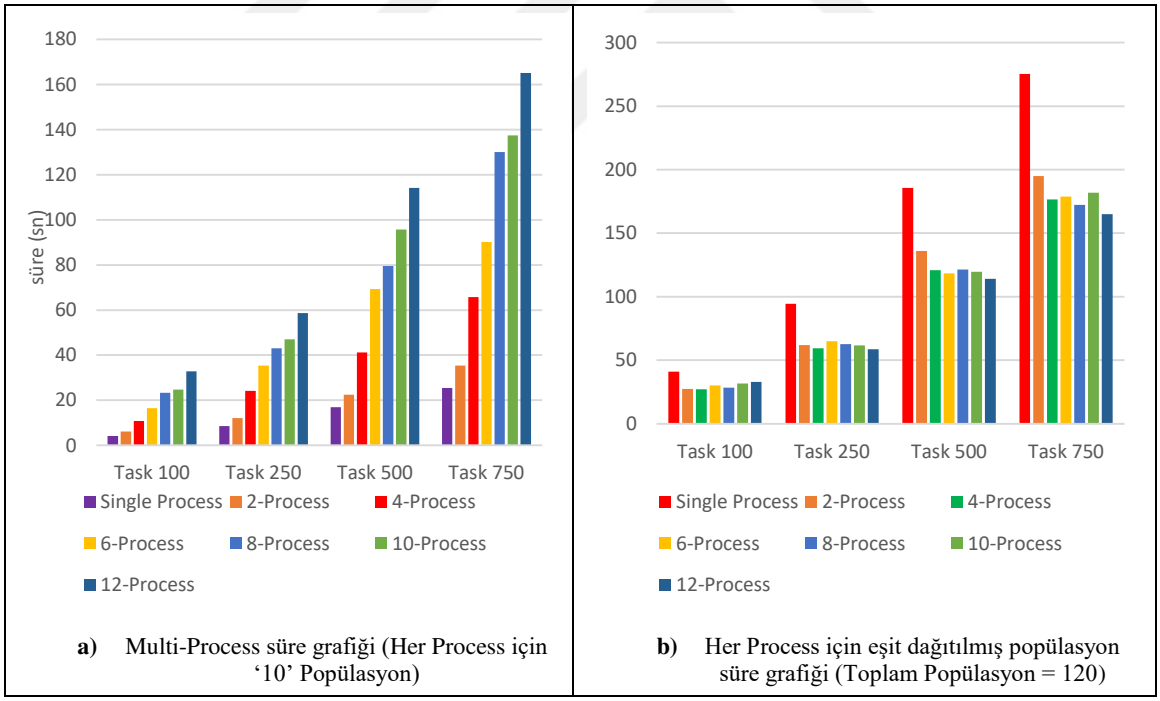
Şekil 4.8. Multi-Thread ve Multi-Process makespan karşılaştırması



a) Multi-Thread süre grafiği (10 Popülasyon)

b) Multi- Process süre grafiği (10 Popülasyon)

Şekil 4.9. Multi Thread ve Multi Process süre karşılaştırması



a) Multi-Process süre grafiği (Her Process için '10' Popülasyon)

b) Her Process için eşit dağıtılmış popülasyon süre grafiği (Toplam Popülasyon = 120)

Şekil 4.10. Multi Process farklı popülasyon senaryoları için süre grafiği

Multi-Process JSO' nun bir önceki deneylerde başarım sağlamasının ardından toplam popülasyon sayısı Process' lere eşit şekilde dağıtılarak iyileşme olup olamayacağı gözlemlenmiştir. Deneyler de daha iyi karşılaştırma yapabilmek için toplam Process sayısına tam bölünebilen popülasyon sayısı (120) belirlenmiştir. Popülasyon sayısı artarken sürenin olumsuz etkilenmesine

rağmen Şekil 4.10.a-b' de görüldüğü üzere toplam popülasyon sayısı '120' olarak belirlenip Process' lere eşit şekilde dağıtılması önemli bir avantaj sağlamayı başarmıştır. Örnek vermek gerekirse Şekil 4.10.a 'da her bir Process' e '10' popülasyon atanmıştır. Şekil 4.10.b' de ise 12-Process için 120 Popülasyon eşit dağıtılmış (her birine 10) ve karşılaştırma yapılmıştır. Sonuçlara bakıldığında Multi-Process çalışma, süre olarak çok iyi başarım sağlamış ama makespan değerinde kısmen benzer sonuçlar elde edilmiştir. Klasik Multi-Process deneylerinin istatistiksel sonuçları Tablo 4.5' te verilmiştir.

Tablo 4.5. Klasik Multi-Process için istatistiksel sonuçlar

Process	Task	Min. ulaşılan iter.	Minimum	Çözüm Süresi	Mean
Single Process x Pop 120	100 Task	246	24.608	41.000	32.030
	250 Task	394	63.416	94.409	85.652
	500 Task	390	138.466	185.750	173.144
	750 Task	317	222.255	275.390	269.341
2- Process x Pop 60	100 Task	425	25.372	27.330	33.516
	250 Task	469	64.652	61.894	84.572
	500 Task	279	136.582	136.112	175.495
	750 Task	478	205.267	194.994	280.243
4- Process x Pop 30	100 Task	374	26.431	27.156	31.733
	250 Task	301	60.706	59.471	78.577
	500 Task	461	129.869	120.965	170.110
	750 Task	480	169.535	176.644	247.321
6- Process x Pop 20	100 Task	419	25.870	30.088	30.128
	250 Task	482	62.082	65.044	79.653
	500 Task	375	111.290	118.404	158.691
	750 Task	498	167.392	178.825	231.829
8- Process x Pop 15	100 Task	496	22.246	28.485	29.146
	250 Task	458	55.768	62.666	71.188
	500 Task	492	100.442	121.436	140.430
	750 Task	419	171.695	172.275	216.920
10- Process x Pop 12	100 Task	468	22.826	31.749	28.391
	250 Task	486	60.079	61.800	72.627
	500 Task	492	102.550	119.664	145.257
	750 Task	482	169.741	181.929	229.831
12- Process x Pop 10	100 Task	288	23.607	32.896	29.308
	250 Task	464	58.176	58.688	72.025
	500 Task	487	103.258	114.144	148.054
	750 Task	481	166.088	165.038	213.483

4.6.1. Önerilen Paralel JSO Modeli

Bir önceki bölüm deneylerinde var olan popülasyonun Process' lere eşit olarak dağıtılması, yürütüm zamanı açısından daha elverişli olduğunu göstermektedir. Bu nedenle bu önerdiğimiz paralel JSO modelde asıl fark popülasyondaki adayların Process' lere eşit şekilde dağıtımı yapılırken birbirine benzer olmayan adayların kullanılmasıdır. Bu kullandığımız teknik C-JSO' da ki benzerlik tekniği ile aynıdır. Tablo 4.6 ve Şekil 4.11.a-b-c-d' de tüm Task senaryoları için

benzerlik oranlarının karşılaştırılması verilmiştir. Deneylerde benzerlik oranları %3, %5, %10 ve %20 seçilmiştir. Tüm Task senaryoları dikkate alındığında Process sayısı arttıkça makespan değerinde iyileşme görülmüştür. Ayrıca en yüksek Process sayısında %3 benzerlik oranının makespan bazında başarım sağladığı görülmüştür.

a) Task 100 için makespan grafiği

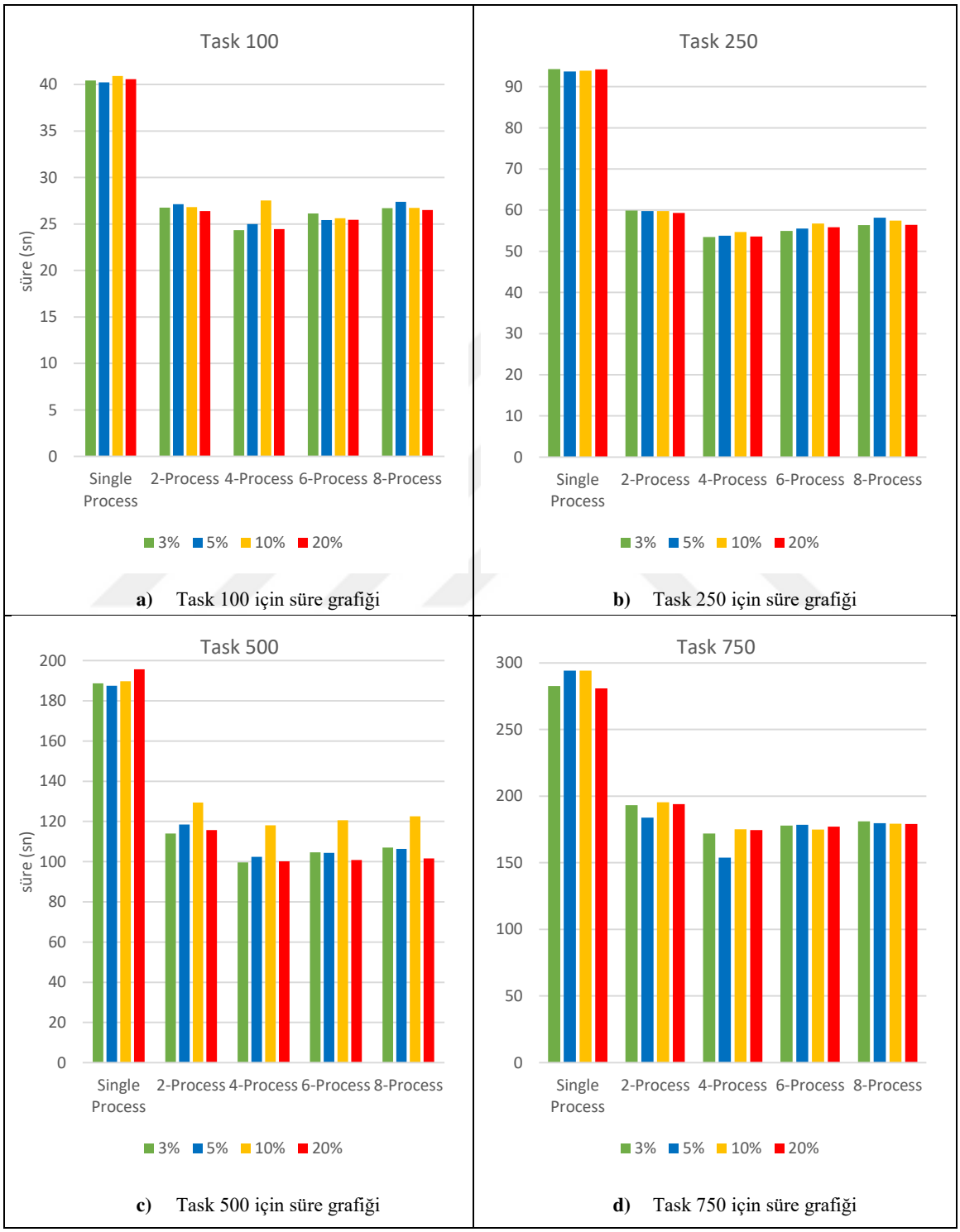
b) Task 250 için makespan grafiği

c) Task 500 için makespan grafiği

d) Task 750 için makespan grafiği

Şekil 4.11. Tüm senaryolar için benzerlik oranları makespan karşılaştırması

Şekil 4.12.a-b-c-d' ye bakıldığında benzerlik oranlarının süre bazında çok etkili olmadığı görülmüştür. Dolayısıyla bundan sonraki deneylerde klasik paralel JSO ve önerilen paralel JSO kıyaslandığında, benzerlik oranı makespan iyileştirme etkisinden dolayı '%3' seçilmiştir.



a) Task 100 için süre grafiği

b) Task 250 için süre grafiği

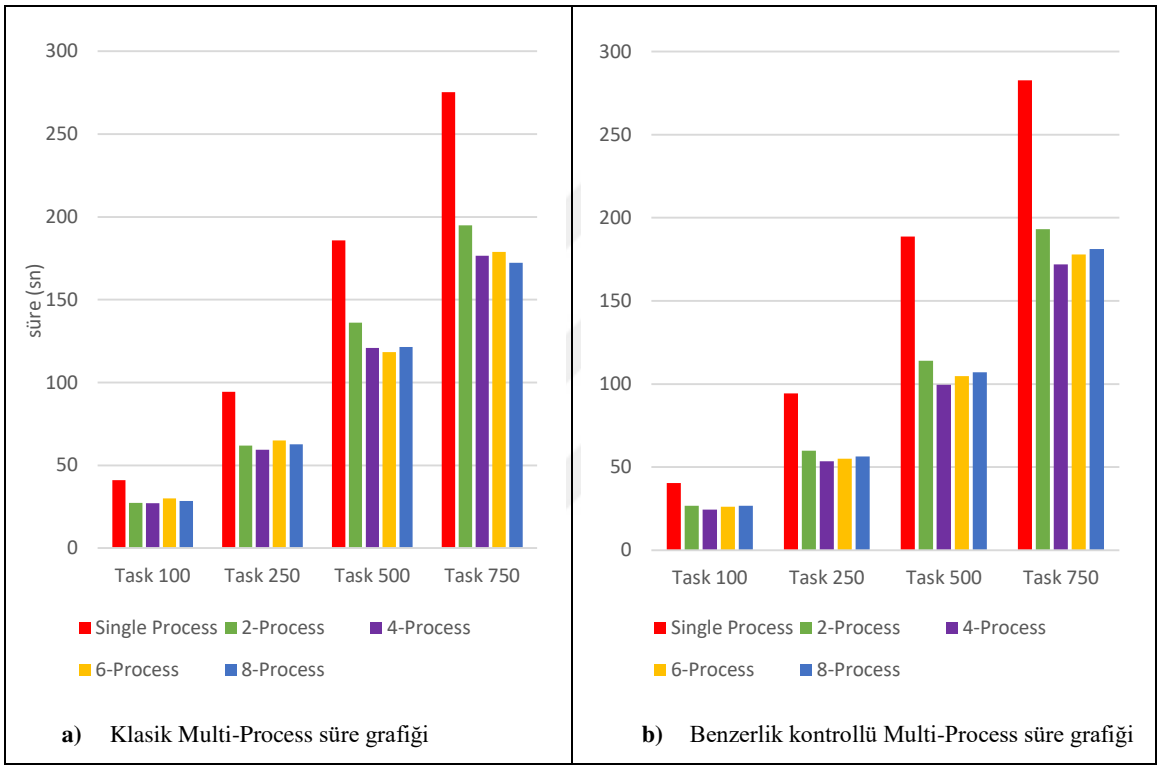
c) Task 500 için süre grafiği

d) Task 750 için süre grafiği

Şekil 4.12. Tüm senaryolar için benzerlik oranları süre karşılaştırması

4.6.2. Önerilen Paralel JSO' nun Karşılaştırmalı Sonuçları

Bu bölümde önerilen paralel JSO (benzerlik kontrollü Multi-Process) ve klasik paralel JSO deneyleri tüm senaryolar için karşılaştırılmıştır. Bu karşılaştırmalar da benzerlik kontrollü Multi-Process, 4-Process' ten sonra yürütüm süresinde olumsuz etkilenmiştir. Bunun nedeni deneylerin yapıldığı bilgisayarın 4-Çekirdekli işlemciye (CPU) sahip olmasıdır. Şekil 4.13.a-b'e bakıldığında klasik yöntem ve geliştirilmiş yöntem 4-Process'te kıyaslandığında yeni yöntem %10 başarım sağlamıştır. Hatta bazı Task senaryolarında bu oran %20 'ye kadar çıkabilmektedir.

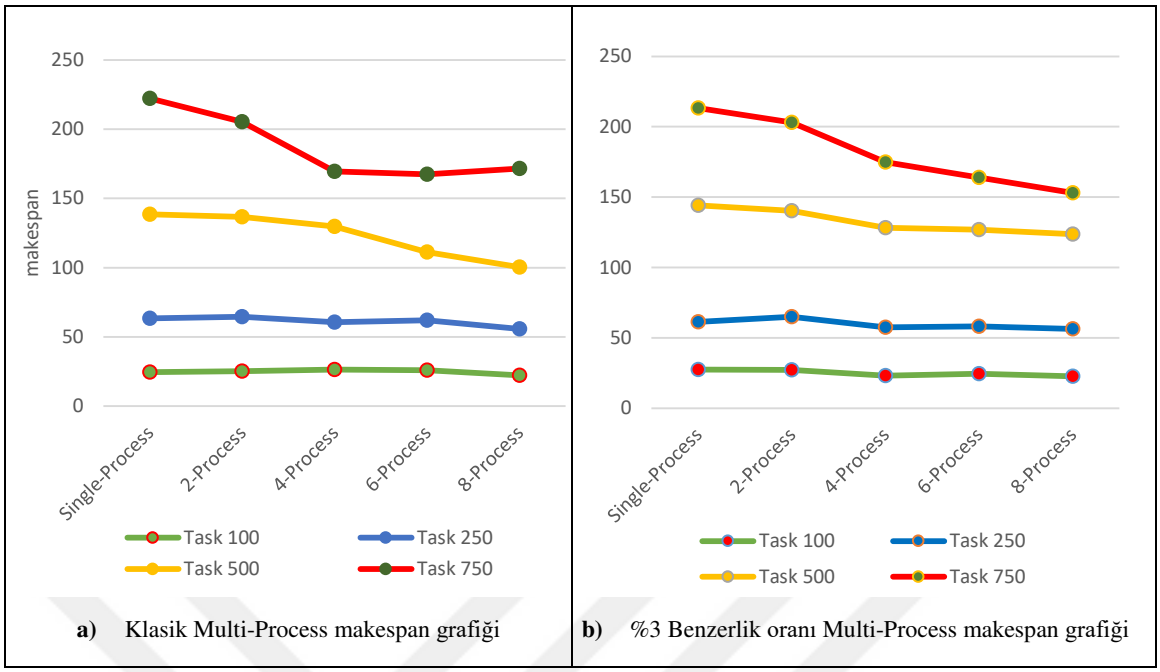


a) Klasik Multi-Process süre grafiği

b) Benzerlik kontrollü Multi-Process süre grafiği

Şekil 4.13. Klasik Multi-Process ve benzerlik kontrollü Multi-Process yürütüm süresi grafiği

Şekil 4.14.a-b' de görüldüğü üzere benzerlik kontrollü Multi-Process, klasik yönteme göre makespan bazında kısmen başarım sağlamıştır. Bu başarımın sebebinin deneylerin yapıldığı bilgisayarın 4-Çekirdekli CPU' ya sahip olmasından dolayı 4-Process' in üzerinde verim düşmesi olduğu varsayılmıştır.



a) Klasik Multi-Process makespan grafiği

b) %3 Benzerlik oranı Multi-Process makespan grafiği

Şekil 4.14. Klasik Multi-Process ve benzerlik kontrollü Multi-Process makespan karşılaştırma grafiği

Tablo 4.6. Benzerlik Kontrollü Multi-Process tüm senaryolar için istatistiksel sonuçlar

Process	Task	Benzerlik Eşik	Minimum	Min. ulaşılan iter.	Çözüm Süresi
Single Process x Pop 120	100 Task	3%	27.399	457	40.424
		5%	25.243	467	40.237
		10%	24.627	404	40.917
		20%	29.338	324	40.564
	250 Task	3%	61.266	288	94.275
		5%	62.369	320	93.724
		10%	69.577	378	93.883
		20%	66.885	486	94.177
	500 Task	3%	144.053	301	188.709
		5%	115.342	423	187.575
		10%	139.731	294	189.733
		20%	125.067	462	195.731
	750 Task	3%	213.317	319	282.598
		5%	215.976	252	294.161
		10%	227.640	287	294.104
		20%	196.849	294	280.909
2 - Process x Pop 60	100 Task	3%	27.206	427	26.743
		5%	23.845	490	27.111
		10%	26.846	289	26.801
		20%	26.049	293	26.392
	250 Task	3%	64.964	440	59.880
		5%	65.294	385	59.751
		10%	62.025	359	59.785
		20%	48.613	379	59.331
	500 Task	3%	140.218	372	113.944
		5%	144.106	459	118.459
		10%	124.470	498	129.477
		20%	116.983	282	115.756
	750 Task	3%	202.997	436	193.221
		5%	191.971	278	183.821
		10%	181.432	290	195.448
		20%	218.521	444	194.056

Tablo 4.6. Benzerlik Kontrollü Multi-Process tüm senaryolar için istatistiksel sonuçlar (Devamı)

Process	Task	Benzerlik Eşik	Minimum	Min. ulaşılan iter.	Çözüm Süresi
4 - Process x Pop 30	100 Task	3%	23.185	418	24.335
		5%	26.193	319	24.983
		10%	22.277	407	27.514
		20%	24.331	388	24437
	250 Task	3%	57.453	486	53.498
		5%	57.479	333	53.783
		10%	54.954	489	54.679
		20%	62.422	317	53.579
	500 Task	3%	128.245	406	99.578
		5%	120.569	344	102.405
		10%	128.583	267	118.081
		20%	124.860	433	100.204
	750 Task	3%	174.928	445	171.942
		5%	185.021	450	153.845
		10%	193.802	459	175.203
		20%	206.960	491	174.531
6 - Process x Pop 20	100 Task	3%	24.512	470	26.138
		5%	23.286	442	25.426
		10%	24.223	313	25.618
		20%	23.271	475	25441
	250 Task	3%	58.229	434	54.978
		5%	56.073	465	55.529
		10%	57.956	399	56.754
		20%	58.543	447	55.823
	500 Task	3%	126.940	495	104.645
		5%	111.692	495	104.367
		10%	108.998	485	120.529
		20%	121.102	421	100.824
	750 Task	3%	163.969	497	177.823
		5%	190.193	497	178.400
		10%	172.270	488	174.949
		20%	194.262	485	177.160
8 - Process x Pop 15	100 Task	3%	22.739	372	26.690
		5%	24.365	381	27.386
		10%	25.017	466	26.721
		20%	23.001	491	26.486
	250 Task	3%	56.280	385	56.339
		5%	59.722	462	58.159
		10%	57.857	492	57.481
		20%	60.435	295	56.438
	500 Task	3%	123.755	496	107.034
		5%	111.895	477	106.300
		10%	106.494	383	122.508
		20%	109.661	492	101.614
	750 Task	3%	152.929	494	181.103
		5%	142.164	488	179.666
		10%	179.173	468	179.322
		20%	182.368	402	179.170

5. SONUÇLAR

Bu tez çalışmasında, bulut bilişimdeki en önemli sorunlardan biri olan görev çizelgeleme süreci üzerine odaklanılmıştır. Sanal makine paylaşımı, güç tasarrufu ve maliyet gibi birçok etken ele alındığında etkili görev çizelgeleme oldukça zordur. Bu tip zor problemlerin çözümünde sıklıkla tercih edilen metasezgisel yaklaşım kullanımı bu tez çalışmasında da kullanılmıştır. Bu sebeple son yılların öne çıkan metasezgisel algoritmalarından olan Denizanası Arama Optimizasyonu (JSO) tercih edilmiştir. İlk olarak bu algoritmanın ne kadar başarılı olduğu iyi bilinen test fonksiyonları sonuçlarına göre değerlendirilmiştir. Yapılan deneylerde JSO' un başarımı ispatlanmış ve ardından bulut ortamlarda görev çizelgeleme problemlerini çözmek için uyarlanmıştır. Ancak bununla birlikte klasik JSO' da diğer metasezgisel algoritmalar gibi lokal minimuma takılma problemine duyarlıdır.

Bu tez çalışması, bulut sistemlerde metasezgisel çizelgelemede iki sorunun cevabını aramıştır. Bunlardan birincisi lokal minimalardan kurtulmak için dinamik popülasyon artışı kullanılabilir mi? İkincisi paralel metasezgisel algoritma yürütümü ile başarım arttırılabilir mi? İlk soru için klasik JSO' nun uyarlanmış bir versiyonu olan C-JSO geliştirilmiştir. Klonlanma tekniği, denizanalarının doğadaki davranışlarından esinlenerek ortaya çıkarılmıştır. C-JSO, durum kontrollü popülasyon artışı ile esnek hale getirebilen mekanizmalar kullanmaktadır. Böylece, bu uyarlanmış meta-sezgisel algoritma, klasik JSO ve C-JSO olmak üzere iki farklı durumda çalışabilmektedir. Yapılan deneylerde C-JSO sonuçları, CloudSim 'in varsayılan çizelgeleme yöntemi ve Karınca Kolonisi Algoritması (ACO) sonuçları ile karşılaştırılmıştır. Hem klasik JSO hem de C-JSO çözümleri standart bulut görev paylaşım yöntemlerine göre daha başarılı sonuçlar vermeyi başarmıştır. Makespan ve çözüm süresi karşılaştırmalarında ve tüm görev sayılarında, C-JSO' nun diğer tüm yöntemlere göre daha başarılı olduğu gözlemlenmiştir. Ayrıca bu tez çalışmasında popülasyon büyümesi sonucu ortaya çıkan performans kaybını düşürmek için paralel JSO modeli de önerilmiştir. Bunun için ilk olarak Multi-Thread ve Multi-Process deneyleri yapılmış ve uygun olan yöntemin Multi-Process olduğu görülmüştür. Bu sonuç, kullanılan programlama dilinin Python olması ve kullandığı GIL mekanizmasından dolayı olduğu anlaşılmıştır. Bu nedenle yeni paralel JSO model Multi-Process temelli geliştirilmiştir. Yapılan deneylerde önerilen yöntemin başarısı özellikle yürütüm süresi açısından net bir şekilde görülmüştür. Böylece tez kapsamında önerilen iki yaklaşım da başarımlarını ispatlamayı başarmıştır.

Bulut sistemler uzun süre farklı problemlerin ortaya çıkacağı bir konu olmaya devam edecektir. Bu nedenle, yazarlar bundan sonraki çalışmalarında bulut sistemlerde ortaya çıkan farklı optimizasyon problemlerinin çözümlerine odaklanacaklardır.

KAYNAKLAR

[1] Houssein, E. H., Gad, A. G., Wazery, Y. M., & Suganthan, P. N. (2021). Task scheduling in cloud computing based on meta-heuristics: review, taxonomy, open challenges, and future trends. Swarm and Evolutionary Computation, 62, 100841.

[2] Abdel-Basset, M. , Laila, A., Sangaiah, A.K. (2018) Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications.

[3] Jui-Sheng, C., & Dinh-Nhat, T. (2021). A novel metaheuristic optimizer inspired by behaviour of jellyfish in ocean, Applied Mathematics and Computation, C. 389, 125535.

[4] Hussain, K., Mohd Salleh, M.N., S. Cheng, Y. S. (2019). Metaheuristic research: a comprehensive survey, Artif. Intell. Rev. 52, 2191–2233.

[5] Tsai, C. W., Huang, W. C., Chiang, M. H., Chiang, M. C., & Yang, C. S. (2014). A hyper-heuristic scheduling algorithm for cloud. IEEE Transactions on Cloud Computing, 2, 236-250.

[6] Kıran, M.S., Gündüz, M., Baykan, Ö.K. (2012). A novel hybrid algorithm based on particle swarm and ant colony optimization for finding the global minimum, Appl. Math. Comput, C. 219, 1515–1521.

[7] Awad, A.I., El-Hefnawy, N.A., Abdel_kader, H.M. (2015). 'Enhanced Particle Swarm Optimization for Task Scheduling in Cloud Computing Environments', Procedia Computer Science, C. 65, 920 – 929.

[8] Alsaidy, S. A., Abbood, A. D., Sahib, M. A. (2020). Heuristic initialization of PSO task scheduling algorithm in cloud computing. Journal of King Saud University-Computer and Information Sciences.

[9] Yildirim, G., & Alatas, B. (2021). New adaptive intelligent grey wolf optimizer based multi-objective quantitative classification rules mining approaches. Journal of Ambient Intelligence and Humanized Computing, 12, 9611–9635. https://doi.org/10.1007/s12652-020-02701-9.

[10] G. Yildirim, Hallac, İ.R., Aydin, G., Tatar, Y. (2015). Running genetic algorithms on Hadoop for solving high dimensional optimization problems, 2015 9th International Conference on Application of Information and Communication Technologies. pp. 12-16, doi: 10.1109/ICAICT.2015.7338506

[11] Li, K., Xu, G., Zhao, G., Dong, Y., Wang, D. (2011). Cloud task scheduling based on load balancing ant colony optimization. In 2011 sixth annual ChinaGrid conference. (pp. 3-9). IEEE.

[12] Belgacem, A., Beghdad-Bey, K., Nacer, H. (2018). Task scheduling optimization in cloud based on electromagnetism metaheuristic algorithm. In 2018 3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS). (pp. 1-7). IEEE.

[13] Yıldırım, S., Yıldırım, G., Alatas, B. (2021). "Anlaşılabilir Sınıflandırma Kurallarının Ayçiçeği Optimizasyon Algoritması ile Otomatik Keşfi", Türk Doğa ve Fen Dergisi, vol. 10, no. 2, pp. 233-241, doi:10.46810/tdfd.976397

[14] Sasikaladevi, N. (2016). Minimum makespan task scheduling algorithm in cloud computing, International Journal of Advances in Intelligent Informatics ISSN: 2442-6571, pp. 123-130.

[15] Liu, C. Y., Zou, C. M., & Wu, P. (2014). A task scheduling algorithm based on genetic algorithm and ant colony optimization in cloud computing. In 2014 13th International Symposium on Distributed Computing and Applications to Business, Engineering and Science, (pp. 68-72). IEEE.

[16] Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., & Buyya, R. (2011). CloudSim: a toolkit for modelling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and experience, 41(1), 23-50.

[17] Abdullahi, M., & Ngadi, M. A. Symbiotic organism search optimization-based task scheduling in cloud computing environment. Future Generation Computer Systems, 56, (2016), 640-650.

[18] Chen, X., Çeng, U., Liu, K., Liu, Q., Liu, J., Mao, Y., Murphy, J. (2020). A WOA-Based Optimization Approach for Task Scheduling in Cloud Computing Systems. Volume: 14, Issue: 3, 3117 – 3128. IEEE.

[19] Dantzig, G.B. (2014). The Nature of Mathematical Programming Archived 2014-03-05 at the Wayback Machine, Mathematical Programming Glossary, INFORMS Computing Society.

[20] Özsağlam, M.Y., & Çunkaş, M. (2008). Optimizasyon Problemlerinin Çözümü için Parçacık Sürü Optimizasyonu Algoritması, Journal of Polytechnic Vol: 11 No: 4 pp.299-305.
[21] Kaya, S., Fığlalı, N. (2016). Çok Amaçlı Optimizasyon Problemlerinde Pareto Optimal Kullanımı, Social Sciences Research Journal, Volume 5, Issue 2, 9718, ISSN: 214775237
[22] Zhao, W., Wang, L., Zhang, Z. (2019). Atom search optimization and its application to solvea hydro geologic parameter estimation problem, Knowl. Based Syst. 163, 283–304.
[23] Aykut, A. (2009). Metasezgisel yöntemlerle uçak çizelgeleme problemi optimizasyonu, Doktora Tezi, Marmara Üniversitesi, 28553986.
[24] Talbi, E.G. (2009) "Metaheuristics: from design to implementation. Hoboken, N.J: John Wiley & Sons. (Pp - 43)
[25] Çelik, K. (2021). Basic Topics in Cloud Computing, Uluslararası Batı Karadeniz Sosyal ve Beşerî Bilimler Dergisi, 5(2): 236-250.
[26] Jakimoski, K., (2016). Security techniques for protecting data in cloud computing, Int. J. Grid Distrib. Comput., 9(1), 49-56.
[27] Voorsluys, W., Broberg, J, Buyya, R. (2011). "Introduction to Cloud Computing.". In R. Buyya, J. Broberg, A.Goscinski. Cloud Computing: Principles and Paradigms. New York, USA: Wiley Press. pp. 1–44. ISBN 978-0-470-88799.
[28] Hamdaqa, M., Livogiannis, T., Tahvildari, L. (2011). A Reference model for developing cloud Applıcatıons, Proceedings of the 1st International Conference on Cloud Computing and Services Science, Noordwijkerhout, Netherlands.
[29] Amalarethinam, D. G. & Beena, T. L. A. (2014). Cloud scheduling-a survey, International Journal of Computer Applications 97(13).
[30] Hashem, I.A.T., Yaqoob, I., Anuar, N.B., Mokhtar, S., Gani, A., Khan, S.U. (2014). The rise of "big data" on cloud computing: Review and open research issues. Information Systems, 47 (2015), 98-115.
[31] Pradhan, A., Bisoy, S. K., & Das, A. (2021). A survey on pso based meta-heuristic scheduling mechanism in cloud computing environment. Journal of King Saud University-Computer and Information Sciences.
[32] Saurav, S. K., & Benedict, S. (2021). A Taxonomy and Survey on Energy-Aware Scientific Workflows Scheduling in Large-Scale Heterogeneous Architecture. In 2021 6th International Conference on Inventive Computation Technologies (ICICT), (pp. 820-826). IEEE.
[33] Bastian, T., Lilley, M.K.S., Beggs, S.E., Hays, G.C., Doyle, TK. (2014). Ecosystem relevance of variable jellyfish biomass in the Irish Sea between years, regions and water types, Estuarine, Coast. Shelf Sci. 149, 302–312.
[34] Fossette, A.C., Gleiss, J., Chalumeau, T. Bastian, C.D. (2015). Armstrong, S. Vandenabeele, M. Karpytchev, G.C. Hays, Current-oriented swimming by jellyfish and its role in bloom maintenance, Curr. Biol. 25, 342–347.
[35] Mariottini, G. L., Pane, L. (2010). Mediterranean jellyfish venoms: a review on scyphomedusae, Mar. Drugs 8, 1122–1152.
[36] Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., & Buyya, R. (2011). CloudSim: a toolkit for modelling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and experience, 41(1), 23-50.
[37] Xu, M., Tian, W. and Buyya, R. (2017). A survey on load balancing algorithms for virtual machines placement in cloud computing. Concurrency and Computation: Practice and Experience, 29(12).
[38] Fang, Y., Wang, F. and Ge, J. (2010). A task scheduling algorithm based on load balancing in cloud computing. In International Conference on Web Information Systems and Mining, Heidelberg.
[39] Hussain, K., Mohd Salleh, M.N.B., Cheng, Naseem, S.R. (2017). Common Benchmark Functions for Metaheuristic Evaluation: A Review. DOI:10.30630/joiv.1.4-2.65.
[40] Li, Jq., Han, Yq. (2020). A hybrid multi-objective artificial bee colony algorithm for flexible task scheduling problems in cloud computing system. Cluster Comput 23, 2483–2499.

[41] Maheswari, P. U., M. Thanka, R., Edwin, E. B. (2019). A hybrid algorithm for efficient task scheduling in cloud computing environment, Int. J. Reasoning-based Intelligent Systems, Vol. 11, No. 2.

[42] Pirozmand, P., Javadpour, A., Nazarian, H. (2022). GSAGA: A hybrid algorithm for task scheduling in cloud infrastructure. https://doi.org/10.1007/s11227-022-04539-8.

[43] Babu, G., Krishnasamy, K. (2013). Task scheduling algorithm based on Hybrid Particle Swarm Optimization in cloud computing environment, Journal of Theoretical and Applied Information Technology 55(1):33-38.

[44] Benlalia, Z., Karim, A., Beni-hssane, A., Khaloufi, H. (2021). New Hybrid Algorithm For Task Scheduling In Cloud Computing, Vol.99. No 24, ss. 1992-8645.

[45] Saif, A., Al-Arasi, R.A. (2018). HTSCC: A Hybrid Task Scheduling Algorithm in Cloud Computing Environment, International Journal Of Computers & Technology, Vol 17(02):7236-7246.

[46] Safi, F., Farinaz, H.E. (2019). Dynamic scheduling applying new population grouping of whales meta-heuristic in cloud computing, The Journal of Supercomputing, 75(3) doi:10.1007/s11227-019-02832-7

[47] Choe, T.Y., Baxodirjonovich, K.N. (2015). Dynamic Task Scheduling Algorithm based on Ant Colony Scheme, International Journal of Engineering and Technology 7(4):1163-1172

[48] Malekloo, M., Kara, N. (2014). Multi-objective ACO virtual machine placement in cloud computing environments. In Globecom Workshops (GC Wkshps), New Jersey.

[49] Rudolph, G. (1990). "Globale Optimierung mit parallelen Evolutionsstrategien". Diplomarbeit. Department of Computer Science, University of Dortmund.

[50] Rosenbrock, H.H. (1960). "An automatic method for finding the greatest or least value of a function". The Computer Journal. 3 (3): 175–184. doi:10.1093/comjnl/3.3.175

[51] Ebbers, M., Kettner, J., O'Brien, W., Ogden, B. (2012). Introduction to the New Mainframe: z/OS Basics. IBM. p. 96. ISBN 978-0-7384-3534-3.

[52] Beazley, D. (2009). "Inside the Python GIL". Chicago: Chicago Python User Group. Retrieved.