

**T.C.
ERCIYES ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI**

**NESNE YÖNELİMLİ YAZILIMLAR VE RESTFUL WEB
SERVİSLER İÇİN YAPAY ARI KOLONİSİ TEMELLİ
YAZILIM TEST YÖNTEMLERİ GELİŞTİRİLMESİ**

**Hazırlayan
Ömür ŞAHİN**

**Danışman
Prof. Dr. Bahriye AKAY**

Doktora Tezi

**Aralık 2020
KAYSERİ**

**T.C.
ERCIYES ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI**

**NESNE YÖNELİMLİ YAZILIMLAR VE RESTFUL WEB
SERVİSLER İÇİN YAPAY ARI KOLONİSİ TEMELLİ
YAZILIM TEST YÖNTEMLERİ GELİŞTİRİLMESİ
(Doktora Tezi)**

**Hazırlayan
Ömür ŞAHİN**

**Danışman
Prof. Dr. Bahriye AKAY**

**Aralık 2020
KAYSERİ**

BİLİMSEL ETİĞE UYGUNLUK

Bu çalışmadaki tüm bilgilerin, akademik ve etik kurallara uygun bir şekilde elde edildiğini beyan ederim. Aynı zamanda bu kural ve davranışların gerektirdiği gibi, bu çalışmanın özünde olmayan tüm materyal ve sonuçları tam olarak aktardığımı ve referans gösterdiğimi belirtirim.

Ömür ŞAHİN

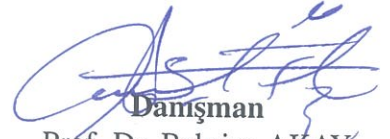
İmza:

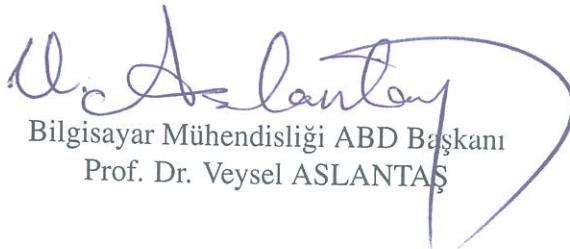


YÖNERGEYE UYGUNLUK

“Nesne Yönelimli Yazılımlar ve RESTful Web Servisler için Yapay Arı Kolonisi Temelli Yazılım Test Yöntemleri Geliştirilmesi” adlı Doktora tezi, Erciyes Üniversitesi Lisansüstü Tez Önerisi ve Tez Yazma Yönergesi’ne uygun olarak hazırlanmıştır.


Tezi Hazırlayan
Ömür ŞAHİN


Danışman
Prof. Dr. Bahriye AKAY


Bilgisayar Mühendisliği ABD Başkanı
Prof. Dr. Veysel ASLANTAŞ

Prof. Dr. Bahriye AKAY danışmanlığında **Ömür ŞAHİN** tarafından hazırlanan “**Nesne Yönelimli Yazılımlar ve RESTful Web Servisler için Yapay Arı Kolonisi Temelli Yazılım Test Yöntemleri Geliştirilmesi**” adlı bu çalışma, jürimiz tarafından Erciyes Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalında **Doktora** tezi olarak kabul edilmiştir.

18/12/2020

JÜRİ :

Danışman : Prof. Dr. Bahriye AKAY
Üye : Prof. Dr. Derviş KARABOĞA
Üye : Doç. Dr. Selçuk ASLAN
Üye : Dr. Öğr. Üyesi Rifat KURBAN
Üye : Dr. Öğr. Üyesi Rüştü AKAY

ONAY :

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

..... / 12 /2020
Prof. Dr. Mehmet AKKURT
Enstitü Müdürü

TEŞEKKÜR

Tez çalışmam boyunca aktardığı bilgi ve tecrübeleri ile yol gösteren, çalışma disiplini ve sahip olduğu bilgi birikimi ile örnek aldığım danışmanım Prof. Dr. Bahriye AKAY’a, tecrübe ve yardımlarını esirgemeyen değerli hocam Prof. Dr. Derviş KARABOĞA’ya teşekkür ederim.

Başta Ahmet Nusret TOPRAK, Ersin KILIÇ ve Tayyip ÖZCAN olmak üzere tüm değerli çalışma arkadaşlarıma teşekkürlerimi sunarım.

Bilgi ve tecrübeleri ile sürekli yol gösteren Necip YILDIZ ve İlhan YILDIZ’a, hayat boyu beni yalnız bırakmayan sevgili Aydan ÇAĞLAR’a teşekkür ederim.

Doktora tez çalışmamı Yurt İçi Lisansüstü Burs Programı kapsamında yürütülen 2211-A Genel Yurt İçi Doktora Burs Programı ile destekleyen TÜBİTAK Bilim İnsanı Destekleme Daire Başkanlığı birimine ve Yüksek Başarım ve Grid Hesaplama Merkezi’nde (TRUBA kaynaklarında) gerçekleştirmiş olduğum kısmi nümerik hesaplamalar sebebiyle TÜBİTAK ULAKBİM’e teşekkür ederim.

Son olarak, üzerimde büyük emeği olan anne, babama, kardeşlerim Onur ve Tülin’e, beni her daim destekleyen sevgili eşim, daimi yoldaşım Gizem’e şükranlarımı sunarım.

NESNE YÖNELİMLİ YAZILIMLAR VE RESTFUL WEB SERVİSLER İÇİN YAPAY ARI KOLONİSİ TEMELLİ YAZILIM TEST YÖNTEMLERİ GELİŞTİRİLMESİ

Ömür ŞAHİN

Erciyes Üniversitesi, Fen Bilimleri Enstitüsü

Doktora Tezi, Aralık 2020

Danışman : Prof. Dr. Bahriye AKAY

ÖZET

Yazılımların yeteri kadar ve doğru bir şekilde test edilememesi neticesinde yazılım projelerinin çoğu başarısız olmaktadır. Yazılım test maliyetinin indirgenmesi ve yazılım kalitesinin artması için test senaryolarının manuel yazılması yerine otomatize edilmesine ihtiyaç duyulmaktadır. Otomatik yazılım test senaryosu üretimi kombinasyonel optimizasyon problemi olarak modellenebilir ve arama algoritmaları ile çözülebilir. Bu tez çalışmasında, nesne yönelimli programlama ve RESTful web servisler için test senaryosu üretimine yönelik yüksek performanslı arama algoritmaları geliştirilmiştir. Tez çalışmasının ilk kısmında, Yapay Arı Koloni algoritması, çaprazlama ve mutasyon operatörleri kullanılarak, nesne yönelimli programlama için test senaryosu üretimine uygun hale getirilmiştir. Daha sonra, keşif ve faydalanma yeteneklerini artırmak amacıyla Modifiye Edilmiş Arşiv Tabanlı Yapay Arı Koloni Algoritması önerilmiştir. Yapılan deneysel çalışmalarda en uygun çaprazlama tekniği belirlenmiş olup, arşiv mekanizmasının etkisi incelenmiştir. Zorluk seviyelerine göre ayrılmış problemler üzerinde çalıştırılan algoritmalar ile elde edilen kapsama miktarına göre algoritmaların performans analizi gerçekleştirilmiştir. İstatistiksel analizlere göre, Modifiye Edilmiş Arşiv Tabanlı Yapay Arı Koloni Algoritması daha az çevrim gerçekleştirerek diğer algoritmalarından daha fazla kapsama miktarına eriştiği görülmüştür.

Tezin ikinci kısmında, RESTful web servisler için test senaryosu üretimi amacıyla Yapay Arı Koloni algoritması uyarlanmış ve eksiklikleri göz önünde bulundurularak Hiper-Kâşif Arı Fazlı Dinamik Yapay Arı Koloni Algoritması olarak isimlendirilen yeni bir Yapay Arı Koloni algoritması önerilmiştir. İlk olarak önerilen algoritmaların kontrol parametreleri belirlenmiş, daha sonra önerilen algoritmalar ile literatürde bilinen algoritmaların kapsadığı hedef sayısı, bulunduğu hata sayısı, üretilen test sayısı ve çalışma zamanı sonuçları istatistiksel analizler kullanılarak kıyaslanmıştır. Elde edilen sonuçlara göre Hiper-Kâşif Arı Fazlı Dinamik Yapay Arı Koloni Algoritması tez çalışmasında ele

alınan diğer algoritmalarından daha yüksek başarımlar elde etmiştir.

Anahtar Kelimeler: Yazılım Testi, Yapay Zeka, Yapay Arı Kolonisi Algoritması, Unit Test, Nesne Yönelimli Programlama, RESTful Web Servis



**ARTIFICIAL BEE COLONY-BASED SOFTWARE TEST SUITE GENERATION
FOR OBJECT-ORIENTED SOFTWARE AND RESTFUL WEB SERVICES****Ömür ŞAHİN****Erciyes University, Graduate School of Natural and Applied Sciences****Ph.D. Thesis, December 2020****Supervisor: Prof. Dr. Bahriye AKAY****ABSTRACT**

Most of the software projects fail because they are not tested sufficiently or correctly. In order to reduce the software testing cost and increase the software quality, writing test cases needs to be automated instead of manual writing. Automating test case generation can be modeled as a combinatorial optimization problem that can be solved by search algorithms. In this thesis, efficient search algorithms were developed for object-oriented software and RESTful web services. In the first part of the thesis, the Artificial Bee Colony algorithm was modified by using crossover and mutation operators to be suitable for test case generation for object-oriented software. Then, a Modified Archive-Based Artificial Bee Colony was proposed in order to increase exploration and exploitation capabilities. In the experimental studies, the most appropriate crossover technique was determined and the effect of the archive mechanism was examined. Performance analysis of the algorithms was carried out according to the amount of coverage on the problems separated by their difficulty levels. According to the statistical analyzes, it was observed that Modified Archive-Based Artificial Bee Colony achieves higher coverage than the other algorithms even it performs fewer cycles.

In the second part of the thesis, the Artificial Bee Colony algorithm was adapted for test case generation of RESTful web services, and a new Artificial Bee Colony algorithm named Dynamic Artificial Bee Colony with Hyper-Scout was proposed by considering its shortcomings. First, the control parameters of the proposed algorithms were tuned and then, the results of the proposed algorithms were compared to those of state-of-the-art algorithms using statistical analysis in terms of the number of covered targets, error findings, number of generated test cases and execution time. According to the results, Dynamic Artificial Bee Colony with Hyper-Scout achieves better performance compared to the other algorithms considered in the thesis.

Keywords: Software Testing, Artificial Intelligence, Artificial Bee Colony Algorithm, Unit Test, Object-Oriented Programming, RESTful Web Service.



İÇİNDEKİLER

NESNE YÖNELİMLİ YAZILIMLAR VE RESTFUL WEB SERVİSLER İÇİN YAPAY ARI KOLONİSİ TEMELLİ YAZILIM TEST YÖNTEMLERİ GELİŞTİRİLMESİ

BİLİMSEL ETİĞE UYGUNLUK SAYFASI	ii
YÖNERGEYE UYGUNLUK SAYFASI	iii
KABUL VE ONAY	iv
TEŞEKKÜR	v
ÖZET	vi
ABSTRACT	viii
İÇİNDEKİLER	x
KISALTMALAR	xiv
TABLolar LİSTESİ	xvi
ŞEKİLLER LİSTESİ	xviii
GİRİŞ	1

1. BÖLÜM YAZILIM TESTİ

1.1. Giriş	5
1.2. Arama Tabanlı Yazılım Testinin Temelleri	9
1.2.1. Temsil Biçimi	14
1.2.2. Amaç Fonksiyonları	15
1.2.2.1. Dal Mesafesi Tabanlı Yaklaşım	16
1.2.2.2. Kontrol Tabanlı Yaklaşımlar	17
1.2.2.3. Karma Yaklaşımlar	18
1.3. Arama Tabanlı Yazılım Testi Hakkında Yapılan Çalışmalar	18

2. BÖLÜM LİTERATÜR TARAMASI

2.1. Nesne Yönelimli Programlama için Test Senaryosu Üretiminde Yapılan Çalışmalar	22
--	----

2.2. RESTful Servisler için Test Senaryosu Üretiminde Yapılan Çalışmalar .	23
--	----

3. BÖLÜM

NESNE YÖNELİMLİ PROGRAMLAMA İÇİN TEST SENARYOSU ÜRETİMİ

3.1. Giriş	26
3.2. Nesne Yönelimli Test Senaryosu Üretimi	27
3.3. Program Analiz Birimi	28
3.4. Temel Yapay Arı Koloni Algoritması	29
3.5. Arşiv Tabanlı Yapay Arı Koloni Algoritması (ABC)	30
3.5.1. Temsil Biçimi	31
3.5.2. Amaç Fonksiyonları	33
3.5.2.1. Metot Kapsamı (Method Coverage)	33
3.5.2.2. Satır Kapsamı (Line Coverage)	33
3.5.2.3. Dal Kapsamı (Branch Coverage)	34
3.5.2.4. Doğrudan Dal Kapsamı (Direct Branch Coverage)	34
3.5.2.5. Çıkış Kapsamı (Output Coverage)	34
3.5.2.6. Zayıf Mutasyon (Weak Mutation)	35
3.5.2.7. İstisna Kapsamı (Exception Coverage)	35
3.5.2.8. Çok Kriterli Amaç Fonksiyonu	35
3.5.3. Başlangıç	36
3.5.4. Çaprazlama ve Mutasyon	36
3.5.5. Görevli Arı Fazı	37
3.5.6. Gözcü Arı Fazı	38
3.5.7. Kâşif Arı Fazı	38
3.5.8. Seçim Stratejisi	39
3.5.9. Arşiv	39
3.5.10. Budama	39
3.6. Modifiye Edilmiş Arşiv Tabanlı Yapay Arı Koloni Algoritması (MABC)	39
3.7. Literatürde Bulunan Algoritmalar	40
3.7.1. Rastgele Arama	40

3.7.2. Standart Genetik Algoritma	40
3.7.3. Monotonik Genetik Algoritma	42
3.8. Deneysel Çalışmalar ve Bulgular	43
3.8.1. Deney Konfigürasyonları	43
3.8.2. Çaprazlama Operatörünün İncelenmesi	48
3.8.3. Arşiv Mekanizmasının Etkinliğinin İncelenmesi	48
3.8.4. Algoritmaların Problem Zorluklarına Göre İncelenmesi	50
3.8.5. İstatistiksel Analiz	54
3.8.6. Yakınsama ve Çeşitlilik Analizi	57
3.8.7. Değerlendirme	58

4. BÖLÜM

RESTFUL WEB SERVİSLER İÇİN TEST SENARYOSU ÜRETİMİ

4.1. Giriş	61
4.2. Önerilen Algoritmaların Bileşenleri	62
4.2.1. Test Senaryosunun Temsil Biçimi	63
4.2.2. Amaç Fonksiyonu	65
4.2.2.1. Komut Kapsamının Elde Edilmesi	66
4.2.2.2. Bytecode Seviyesinde Dal Kapsamının Elde Edilmesi	66
4.2.2.3. HTTP Durum Kodlarının Elde Edilmesi	67
4.2.2.4. Test Hedeflerinin Bilgilerinin Sunulması	68
4.2.3. Doğrulayıcı (Oracle)	69
4.2.4. Yeni Çözüm Üretimi	72
4.2.5. Arama Operatörü	76
4.3. RESTful Web Servisler için Test Senaryosu Üretiminde ABC Algoritması	78
4.4. Hiper-Kâşif Arı Fazlı Dinamik Yapay Arı Koloni Algoritması (DABC-HS)	80
4.4.1. Arşiv Bloğu	81
4.4.2. Başlangıç	84
4.4.3. Görevli Arı Fazı	84

4.4.4. Gözcü Arı Fazı	85
4.4.5. Kâşif Arı Fazı	87
4.4.6. Hiper-Kâşif Arı Fazı	87
4.5. Literatürde Bulunan Algoritmalar	88
4.5.1. Tam Test Paketi	88
4.5.2. Çok Hedefli Sıralama Algoritması	88
4.5.3. Çoklu Bağımsız Hedef Algoritması	91
4.6. Deneysel Çalışmalar ve Bulgular	93
4.6.1. ABC ve DABC-HS Algoritmalarının Kontrol Parametre Analizi	93
4.6.2. Literatürdeki Algoritmalarla Karşılaştırma	96
4.6.3. Değerlendirme	108
5. BÖLÜM	
TARTIŞMA, SONUÇ ve ÖNERİLER	
5.1. Tartışma ve Sonuçlar	110
5.2. Öneriler	113
KAYNAKLAR	115
EKLER	134
ÖZGEÇMİŞ	140

KISALTMALAR

ABC	: Yapay Arı Koloni Algoritması
ACO	: Karınca Koloni Algoritması
API	: Uygulama Programlama Arayüzü
CBO	: Nesneler arası bağılılık
DABC-HS	: Hiper-Kaşif Arı Fazlı Dinamik Yapay Arı Koloni Algoritması
DE	: Evrimsel Gelişim Algoritması
DIT	: Kalıtım Ağacı Derinliği
EDAs	: Dağıtım Algoritmalarının Tahmini
FA	: Ateş Böceği Algoritması
GA	: Genetik Algoritma
HTTP	: Hiper Metin Transferi Protokolü
JSON	: JavaScript Nesnesi Gösterimi
JVM	: Java Sanal Makinası
loc	: Satır Sayısı
MABC	: Modifiye Edilmiş Arşiv Tabanlı Yapay Arı Koloni Algoritması
MGA	: Monotonik Genetik Algoritma
MIO	: Çoklu Bağımsız Hedef Algoritması
MOSA	: Çok Hedefli Sıralama Algoritması
NOC	: Çocuk Sayısı
NOF	: Alan Sayısı
NOM	: Metot Sayısı
NOPF	: Erişilebilir Alan Sayısı
NOPM	: Erişilebilir Metot Sayısı
NOSF	: Statik Alan Sayısı
NOSI	: Statik Çağrı Sayısı
NOSM	: Statik Metot Sayısı
PSO	: Parçacık Sürü Optimizasyoun
QoS	: Servis Kalitesi
REST	: Temsili Durum Aktarımı
RFC	: Benzersiz Metot Çağrı Sayısı

RS	: Rastgele Arama
SA	: Isıl İşlem Algoritması
SBST	: Arama Tabanlı Yazılım Testi
SGA	: Standart Genetik Algoritma
SLA	: Hizmet Seviyesi Anlaşması
SOAP	: Basit Nesne Erişim Protokolü
SP	: Tek Noktalı Çaprazlama
SPF	: Sabit Tek Noktalı Çaprazlama
SPR	: Bağlı Tek Noktalı Çaprazlama
STGP	: Strongly-Typed Genetik Programlama
SUT	: Test Edilen Sistem
TLBO	: Öğretme-öğrenme Esaslı Optimizasyon Algoritması
URI	: Tekdüzen Kaynak Tanımlayıcı
WMC	: Ağırlıklandırılmış Sınıf Metodu Metriği
WTS	: Tam Test Senaryosu
WWW	: Dünya Çapında Ağ
XML	: Genişletilebilir İşaretleme Dili

TABLOLAR LİSTESİ

Tablo 1.1.	Tracey'nin önermiş olduğu dal mesafesi fonksiyonu.	17
Tablo 3.1.	Çaprazlama operatörlerinin etkisini incelemek için kullanılan sınıfların özellikleri	45
Tablo 3.2.	Her bir seviyede sınıfların metriklerinin tanımlayıcı istatistik değerleri. Ortalama $\pm$ standart sapma değerleri ilk satırda, alt satırda parantez içinde ise ortanca değerleri verilmiştir.	46
Tablo 3.3.	Tablo 3.2'de verilen metriklerin açıklamaları.	47
Tablo 3.4.	Farklı çaprazlama tekniklerinin ABC algoritması üzerindeki etkileri. Ortalama $\pm$ standart sapma değerleri ilk satırda, alt satırda parantez içinde ise ortanca değerleri verilmiştir.	49
Tablo 3.5.	ABC algoritmasının arşivli ve arşivsiz versiyonlarının karşılaştırılması. Ortalama $\pm$ standart sapma değerleri ilk satırda, alt satırda parantez içinde ise ortanca değerleri verilmiştir.	51
Tablo 3.6.	MABC ve ABC algoritmalarının literatürdeki popüler algoritmalar ile kıyaslanması. Ortalama $\pm$ standart sapma değerleri ilk satırda, alt satırda parantez içinde ise ortanca değerleri verilmiştir.	53
Tablo 3.7.	Proje bazlı elde edilen sonuçların Dal Kapsamı metriğine göre 20 bağımsız koşmanın ortalaması.	54
Tablo 3.7.	Proje bazlı elde edilen sonuçların Dal Kapsamı metriğine göre 20 bağımsız koşmanın ortalaması.	55
Tablo 3.7.	Proje bazlı elde edilen sonuçların Dal Kapsamı metriğine göre 20 bağımsız koşmanın ortalaması.	56
Tablo 3.8.	Sütundaki bir algoritmanın satırdaki algoritmaya Dal Kapsamı sonuçlarına uygulanan Wilcoxon testi ($p\text{-degeri} \leq 0.05$) sonucunda üstünlük sağladığı sınıf sayısı İİ: İstatistiksel olarak daha iyi sınıf sayısı, İK: İstatistiksel olarak daha kötü sınıf sayısı.	57
Tablo 3.9.	Her bir algoritmaya ait Dal Kapsamı sonuçlarının ortalama Vargha-Delaney etki büyüklüğü $\hat{A}_{12}$ değeri.	58

Tablo 4.1.	Dal mesafesi fonksiyonu.	67
Tablo 4.2.	ABC and DABC-HS algoritmaları için araştırılan kontrol parametre setleri.	94
Tablo 4.3.	Her bir parametre için ABC algoritması ile elde edilen sonuçların göreceli sıralama değeri. En iyi ayar seti kalın puntolarla gösterilmiştir. GS: Göreceli Sıralama	95
Tablo 4.4.	Her bir parametre için DABC-HS algoritması ile elde edilen sonuçların göreceli sıralama değeri. En iyi ayar seti kalın puntolarla gösterilmiştir. GS: Göreceli Sıralama	96
Tablo 4.5.	Önerilen yöntemlerin analizinde kullanılan SUT'ların özellikleri. . .	97
Tablo 4.6.	Deneylerde elde edilen ortalama kapsama miktarı. Daha iyi sütununda, Mann-Whitney-Wilcoxon U-testi sonucu $p < 0.05$ olan algoritmalar verilmiştir. Algoritma isimlerinin yanında parantez içerisinde ise Vargha-Delaney etki büyüklüğü ($\hat{A}_{12}$) verilmiştir.	98
Tablo 4.7.	Algoritmaların sıralama tabanlı ortalama performansları.	99
Tablo 4.8.	5xx hata sayılarının ortalama değerleri. DABC-HS pivot algoritmadır ve DABC-HS ile diğer algoritmalar arasındaki Vargha-Delaney etki büyüklüğü $\hat{A}_{12}$ alt satırda, parantez içerisinde verilmiştir. Mann-Whitney-Wilcoxon U-testi sonucu $\alpha = 0.05$ güven aralığında anlamlı ise kalın puntolarla yazılmıştır.	100
Tablo 4.9.	Üretilen test sayılarının ortalama değerleri. DABC-HS pivot algoritmadır ve DABC-HS ile diğer algoritmalar arasındaki Vargha-Delaney etki büyüklüğü $\hat{A}_{12}$ alt satırda, parantez içerisinde verilmiştir. Mann-Whitney-Wilcoxon U-testi sonucu $\alpha = 0.05$ güven aralığında anlamlı ise kalın puntolarla yazılmıştır.	102
Tablo 4.10.	Çalışma zamanının ortalama değerleri. DABC-HS pivot algoritmadır ve DABC-HS ile diğer algoritmalar arasındaki Vargha-Delaney etki büyüklüğü $\hat{A}_{12}$ alt satırda, parantez içerisinde verilmiştir. Mann-Whitney-Wilcoxon U-testi sonucu $\alpha = 0.05$ güven aralığında anlamlı ise kalın puntolarla yazılmıştır.	103

ŞEKİLLER LİSTESİ

Şekil 1.1.	Genel arama tabanlı yazılım test senaryosu/verisi üretim şeması	11
Şekil 1.2.	İkinci dereceden denklem kökleri bulan kod parçası ve CFG gösterimi.	13
Şekil 3.1.	Örnek bir sınıf ve bu sınıfa ait test senaryosu.	27
Şekil 3.2.	Genel arama tabanlı test senaryosu üretim şeması.	28
Şekil 3.3.	Çaprazlama operatörünün çalışmasına örnek.	37
Şekil 3.4.	Mutasyon operatörünün çalışmasına örnek.	37
Şekil 3.5.	WMC değerlerinin histogram grafiği.	47
Şekil 3.6.	Algoritmalar tarafından üretilen kapsama miktarlarının her bir seviyeye göre sonuçları.	52
Şekil 3.7.	Global en iyi çözümün yakınsaması	59
Şekil 3.8.	Popülasyon çeşitlilik oranı	59
Şekil 4.1.	Bir kaynağa ait POST sorgusunun örnek Swagger tanımı.	65
Şekil 4.2.	Örnek bir if-else bloğu.	67
Şekil 4.3.	Test senaryosu çalıştırıldıktan sonra üretilen örnek JSON formatında rapor.	70
Şekil 4.4.	Birim test örneği.	71
Şekil 4.5.	Nesne yönelimli test senaryosu örneği	73
Şekil 4.6.	HTTP çağrı örneği	73
Şekil 4.7.	Rastgele örnekleme ile oluşturulan test senaryosu.	76
Şekil 4.8.	Akıllı örnekleme ile oluşturulan test senaryosu.	76
Şekil 4.9.	Yedi HTTP isteğinden oluşan örnek test senaryosu.	77
Şekil 4.10.	Mutasyona uğramış test senaryosu.	77
Şekil 4.11.	DABC-HS algoritmasının genel yapısı.	81

Şekil 4.12.	Arşiv mekanizmasının genel yapısı.	83
Şekil 4.13.	Farklı arama algoritmalarının catwatch problemi üzerinde elde etmiş oldukları sonuçların grafiksel gösterimi.	105
Şekil 4.14.	Farklı arama algoritmalarının Features-service problemi üzerinde elde etmiş oldukları sonuçların grafiksel gösterimi.	105
Şekil 4.15.	Farklı arama algoritmalarının Proxyprint problemi üzerinde elde etmiş oldukları sonuçların grafiksel gösterimi.	106
Şekil 4.16.	Farklı arama algoritmalarının Rest-ncs problemi üzerinde elde etmiş oldukları sonuçların grafiksel gösterimi.	106
Şekil 4.17.	Farklı arama algoritmalarının Rest-news problemi üzerinde elde etmiş oldukları sonuçların grafiksel gösterimi.	107
Şekil 4.18.	Farklı arama algoritmalarının Rest-scs problemi üzerinde elde etmiş oldukları sonuçların grafiksel gösterimi.	107
Şekil 4.19.	Farklı arama algoritmalarının Scout-api problemi üzerinde elde etmiş oldukları sonuçların grafiksel gösterimi.	108

GİRİŞ

Oldukça uzun süredir yazılım geliştiriliyor olsa da projelerin başarılı bir şekilde sonlandırılmasında hâlâ çeşitli başarısızlıklar ortaya çıkmaktadır [1]. Standish Group'un 1994'te yayınlamış olduğu raporda, yazılım projelerinin %31'inin tamamlanmadan önce iptal edildiği, %53'üne itiraz edildiği ve proje maliyetinin ilk tahminlere göre %180 daha fazla olduğu belirtilmiştir [2]. Yine Standish Group tarafından 2006 yılında yayınlanan rapora göre projelerin yalnızca %35'inin istenen zamanda, talep edilen gereksinimleri karşılayacak şekilde tamamlandığı, %19'unun ise tamamlanamadan iptal edildiğini göstermiştir. %46'lık büyük çoğunlukta bulunan projeler ise geç teslim edilmiş, bütçeyi aşmış veya gereksinimleri sağlayamamıştır [3]. Glass'ın yapmış olduğu çalışmada, çoğunluğu büyük projeler olan on altı başarısızlıkla sonuçlanan projeyi incelemiş ve yalnızca yönetsel sorunlar değil teknik problemler sebebiyle de projelerin başarısız olduğunu belirtmiştir [4].

Projelerin başarısız olmasına sebep olan teknik sebeplerden biri yazılımların yeteri kadar veya doğru test edilememesidir. Dünyada yaşanan gelişmelerle birlikte, yazılımların kapsamı artmıştır ve çok sayıda modül içerir hale gelmiştir. Her bir modül doğru şekilde test edilmezse yazılım maliyeti oldukça artmaktadır. Ancak, yazılım testi de oldukça fazla emek gerektirmektedir ve bu testin maliyeti toplam maliyetin neredeyse yarısına eşittir [5]. Projelerin doğru bir şekilde test edilmemesi durumunda yaşanacak sorunlar yazılımın türüne göre değişmektedir ve hatalar barındıran yazılımların piyasaya sürülmesi durumunda, şirketler itibar kaybına uğramaktadır. Uygulamalar piyasaya sürülmeden önce yalnız geliştirilme aşamasında değil, uygulama piyasaya sürüldükten sonra da bakım faaliyetleri sırasında eklenen yeni bileşenlerin veya değiştirilmiş kısımların test edilmesi gerekmektedir. Bu sebeple, yazılım testi hem yazılım geliştirmede, hem de bakım aşamalarında oldukça önemli bir rol oynamaktadır. Çünkü yazılım testleri, yazılımda gerçekleştirilen değişiklikler altında da hedeflenen davranışları (output specifications)

sergiliyor olduğundan emin olunmasını sağlamaktadır.

Yazılım testinin gerçekleştirilebilmesi için test senaryolarının yazılması gerekmektedir. Genellikle bu test senaryoları test uzmanları tarafından el ile yazılmaktadır. Test uzmanları özellikleri incelemeli, neyi test edeceklerini ve test senaryosu yazarken hangi stratejiyi kullanacaklarına karar vermeleri gerekmektedir. Bu sebeple yazılım testinin yazılması oldukça maliyetli, zor ve hataya açık bir süreçtir [6, 7]. Araştırmacılar, el ile test senaryosu oluşturma maliyetini düşürmek ve aynı zamanda test süreçlerinin güvenilirliğini artırmak için bu süreci otomatik hale getirmeye çalışmışlardır. Araştırmalar sonucunda, yazılım test senaryosu üretiminin bir optimizasyon problemi olarak ele alınıp, arama algoritmaları ile çözülmesini temel alan Arama Tabanlı Yazılım Testi (Search-based Software Testing, SBST) tekniği popüler hale gelmiştir ve bu alanda yapılan çalışmaların büyük bir çoğunluğu SBST alanındadır.

Bu tez çalışmasında, nesne yönelimli programlama ve RESTful web servisler için test senaryosu üretimine yönelik yüksek performanslı algoritmalar geliştirilmiştir. Yazılımların karmaşıklığı ve boyutlarının artmasıyla yapısal programlama teknikleri yerine gürbüz mimarilerin tasarlanabildiği nesne yönelimli programlama gibi paradigmlar oldukça popüler hale gelmiştir. Ancak nesne yönelimli yaklaşımın kalıtım, kapsülleme, çok-biçimlilik, soyutlama, bilgi gizleme gibi özellikleri de test süreçlerini zorlaştırmaktadır [8]. Bu nedenle manuel test senaryosu üretim maliyeti daha da artmaktadır. Bir başka yazılım geliştirme yaklaşımı da mikroservis mimarisidir. Son zamanlarda oldukça geniş bir kitleye hitap eden büyük uygulamaların monolitik mimari ile tasarlanması bakım maliyetlerini artırdığından monolitik yapıları yüksek düzeyde kolay bakım yapılabilir servisler topluluğuna dönüştüren mikroservis mimarisi oldukça popüler olmuştur. Ancak mikroservis mimarisinin Otomatik Hizmet Keşfi (automated service discovery), Aşırı Geç Bağlanma (ultra-late binding) gibi özellikleri entegrasyon testini engellerken, kullanım başına maliyet birim testini ve Hizmet Kalitesi (Quality Of Service, QoS) testini etkilemektedir [9, 10]. Bu nedenle yazılım testi oldukça maliyetlidir. Maliyeti düşürmenin en kolay yollarından biri ise yazılım test senaryolarının bir araç yardımıyla otomatik olarak üretilmesidir.

Bu tez çalışmasında ilk olarak Yapay Arı Koloni (Artificial Bee Colony, ABC)

algoritması, çaprazlama ve mutasyon operatörleri ile kombinasyonel problem olan nesne yönelimli programlama için test senaryosu üretimine uygun hale getirilmiştir. Ardından literatürde sıkça kullanılan çaprazlama modellerinin etkinliği SF110 corpus problem seti içinden rastgele seçilmiş 43 sınıf kullanılarak incelenmiştir. Sonraki üç deney için SF110 corpus problem setinden seçilmiş olan 6569 sınıf Ağırlıklandırılmış Sınıf Metodu Metriği (Weighted Method per Class, WMC) metriği kullanılarak 5 farklı zorluk seviyesine göre bölünmüştür. Bu farklı zorluktaki sınıflar kullanılarak optimize edilmiş bireylerin tutulduğu arşiv mekanizmasının etkisi incelenmiştir. Sonraki deneyde keşif ve faydalanma yeteneklerini artırmak amacıyla tez kapsamında önerilen Modifiye Edilmiş Arşiv Tabanlı Yapay Arı Koloni Algoritması (Modified Archive-Based Artificial Bee Colony Algorithm, MABC) algoritmasının etkisi literatürde sıkça kullanılan algoritmalar olan Standart Genetik Algoritma (Standard Genetic Algorithm, SGA), Monotonik Genetik Algoritma (Monotonic Genetic Algorithm, MGA) ve Rastgele Arama (Random Search, RS) kullanılarak incelenmiştir. Son olarak gerçekleştirilen deneylerin istatistiksel testlerle doğrulanması için seçilmiş olan 180 sınıf üzerinde 20 sefer MABC, MGA, SGA ve RS algoritmaları çalıştırılarak dal kapsamı metriğine göre kıyaslanmıştır. Wilcoxon test sonuçları ve Vargha-Delaney etki büyüklüğü $\hat{A}^{12}$ değerleri ile birlikte tanımlayıcı istatistik ve sıralama sonuçları da verilmiştir.

Tez çalışmasının ikinci bölümünde ise RESTful web servisler için test senaryosu üretimi amacıyla ABC algoritması uyarlanmış ve eksiklikleri göz önünde bulundurularak Hiper-Kâşif Arı Fazlı Dinamik Yapay Arı Koloni Algoritması (Dynamic Artificial Bee Colony with Hyper-Scout, DABC-HS) olarak isimlendirilen yeni bir ABC algoritması önerilmiştir. Deneylerin ilk kısmında ABC ve DABC-HS algoritmalarının kontrol parametre ayarlaması gerçekleştirilmiştir. Daha sonra bu algoritmaların performansını; kapsanan hedefler, bulunan hatalar, üretilen testler ve çalışma zamanı açısından incelemek için literatürde sıkça kullanılan problemler olan catwatch, features-service, proxyprint, rest-ncs, rest-news, rest-scs ve scout-api isimli Test Edilen Sistemler (System Under Test, SUT) üzerinde test senaryosu üretimi gerçekleştirilmiştir. Elde edilen sonuçlar, literatürdeki algoritmalarından olan RS, Tam Test Paketi (Whole Test Suite, WTS), Çoklu Bağımsız Hedef Algoritması (Many Independent Objective, MIO) ve Çok Hedefli Sıralama Algoritması (Many-Objective Sorting, MOSA) ile alınan sonuçlar

karşılaştırılmıştır.

Tezin Organizasyonu

Bu tez çalışması şu şekilde organize edilmiştir:

- Yazılım testi ve SBST ilgili genel bilgiler Bölüm 1’de verilmiştir.
- Bölüm 2’de SBST, nesne yönelimli programlama için test senaryosu üretimi ve RESTful web servisler için test senaryosu üretimi ile ilgili ayrıntılı literatür taraması verilmiştir.
- Bölüm 3’te nesne yönelimli programlama için test senaryosu üretiminin temelleri sunulmuştur. ABC algoritmasının uyarlanması ve önerilen MABC algoritmasının ayrıntıları bu bölümde verilmiştir. Ayrıca ABC ve MABC algoritmaları ile gerçekleştirilen deneysel çalışmalar sunulmuştur.
- RESTful web servisler için test senaryosu üretiminin ayrıntıları Bölüm 4’te verilmiştir. ABC algoritmasının RESTful web servisler için test senaryosu üretimi ve önerilen DABC-HS algoritmasının ayrıntıları bu bölümde sunulmuştur. Ayrıca bu algoritmalar ile gerçekleştirilen deneysel çalışmalar da bu bölümde verilmiştir.
- Tez çalışması ile ilgili tartışma, sonuçlar ve öneriler Bölüm 5’te verilmiştir.

1. BÖLÜM

YAZILIM TESTİ

1.1. Giriş

Yazılım bütçesinde oldukça büyük bir paya sahip olan test ve bakım süreçlerinin etkin yürütülmesi oldukça önemlidir. Bu süreçlerin etkinliğini ise yazılan senaryolar ve bu senaryoların yürütülmesinde kullanılan test verileri belirlemektedir. Yazılım testinde, test verisinin/senaryosunun belirlenmesi oldukça zorlu ve emek gerektiren eylemlerden biridir [11–13]. Yazılımların boyutlarının ve karmaşıklığının artması, masaüstü, mobil ve web gibi çeşitli platformlar için pek çok farklı programlama dili ile gerçekleştirilen yazılımların test edilmesini oldukça zorlu bir süreç haline gelmiştir. Uygulamaların test edilebilmesi için gerekli olan test senaryolarının oluşturulması, sürücü metotlar (driver) altında test edilen fonksiyonlara gönderilecek uygun verilerin üretilmesini sağlayan karar verilemeyen (undecidable) bir problemdir ve doğrulayıcı (oracle) hedefler ile çıktıların uyumluluğunu doğrular. Saydam kutu testlerinde, bir doğrulayıcı olmadığı durumlarda incelenen problem için en az sayıda senaryo ile belirli bir kapsam metriğinin maksimize edilmesi hedeflenir. Kısa ancak etkin test paketi (test suite) üretimi uzun uğraşlar ve uzman tecrübesi gerektirir. Farklı platformlar ve programlama dilleri için test verisi/senaryosu (giriş ve beklenen çıktı çiftleri) üretiminde kullanılan araçlar şu şekilde gruplanabilir [14]:

- Sembolik çalıştırma (Symbolic Execution) ve program yapısal kapsam testi (program structural coverage testing)
- Model tabanlı test (Model-based testing)
- Kombinasyonel test (Combinatorial testing)

- Adaptif rastgele test (Adaptive random testing)
- Arama tabanlı test (Search-based testing)

Aynı zamanda mutasyon testi (mutation testing), başkalaşım testi (metamorphic testing), belirtim tabanlı test (specification-based testing), bulanıklaştırma ve veri mutasyonu testi (fuzzing and data mutation testing) gibi test verisi üretmeye yönelik teknikler de bulunmaktadır [14].

Sembolik çalıştırma, King [15] tarafından önerilen, programının iç yapısının bilinmesini gerektiren saydam kutu test yöntemlerinden biridir. Giriş olarak somut veriler yerine sembolik veriler kullanılarak programın davranışı analiz edilir. Yüksek hesaplama maliyeti nedeniyle bilgisayarların güçlenmesi sonrasında sembolik çalıştırma popüler olmuş ve üzerinde pek çok araştırma yapılmıştır. Bu araştırmaların temel amacı kapsama miktarını artırmak ve uygulamadaki hataları ortaya çıkarmak olmuştur [16,17]. Sembolik çalıştırma ile test verisi üretimi gerçekleştiren bazı araçlar bulunmaktadır. Java için Symbolic Pathfinder [18], JCUTE [19], JFuzz [20] ve LCT [21] isimli yazılımlar bulunmaktadır. C dili için ise CUTE [22], Klee [16], S2E [23] araçları geliştirilmiştir. Son olarak, Pex [24] .NET dilleri için geliştirilmiş bir uygulamadır. Son yıllarda Z3 [25], Yices [26], STP [27] gibi birçok güçlü kısıtlama çözücülerin geliştirilmesi ile çalışmalara devam ediliyor olsa da günümüz programlarının boyutlarının oldukça büyük olması nedeniyle tek başına verimli sonuçlar üretmesi oldukça güç olmaktadır. Hesaplama maliyetinin yüksek oluşundan, yol infilakı (path explosion), yol farklılığı (path divergence), ve kompleks kısıtlardan dolayı gerçek dünya yazılımlarına uygulanması güçtür [14].

Model tabanlı test verisi üretimi programın davranışını modellemektir. Girişlerin formatı sonlu durum makinaları, UML durum diyagramları, petri ağlarına veya Kripke yapılarına dayalı temsiller olabilmektedir. Öncelikle SUT için bir model oluşturulur ve model ile girişe bağlı olarak çıkış üretilmektedir. Bu yöntem kara kutu testini temel almaktadır. Bu yöntem ile geliştirilmiş çeşitli araçlar bulunmaktadır. The Conformiq Designer [28] 2006 yılında piyasaya sürülmüştür araçtır. Test için UML diyagramından faydalanmaktadır. Ayrıca The Smartesting CertifyIt [29] ve Spec Explorer [30] isimli yazılımlar piyasada bulunmaktadır. 2015 yılında geliştirilmiş Momut, UML diyagramları

üzerinde model tabanlı mutasyon testleri üreten bir araçtır [31]. Sonlu durum makinaları gerçek dünya problemlerini ifade etmekte yetersiz iken genişletilmiş sonlu durum makinalarının (extended finite state machines) boyutu çok büyük olmaktadır. Bu nedenle pratikte kullanımı çok kısıtlıdır.

Kombinasyonel test üretiminde [32] genellikle bazı parametrelerin programlarda hatalara sebebiyet verdiği kabulünden yola çıkılarak ikili, üçlü, dörtlü gibi parametre veya konfigürasyon kombinasyonları, uzaydaki örnek giriş parametreleri olarak seçilip bu parametreler ile programın davranışı incelenmektedir. Kombinasyonel testlerin temeli deneysel tasarım (Experimental Design) adı verilen istatistik alanından gelmektedir [33, 34]. 1930'lu yıllarda R.A. Fisher [34] bağımsız değişkenleri sistematik bir şekilde birleştiren bir araç tanımlamıştır. 1985'te Mandl bu fikri derleyicinin parametre değerlerinin kombinasyonlarını örneklemek için kullanmıştır [35]. Amerikan Ulusal Standartlar ve Teknoloji Enstitüsü (NIST) tarafından 2009 yılında Automated Combinatorial Testing for Software¹ (ACTS) isimli bir araç çıkarılmıştır. Bu araç geliştirilmiş algoritmalarla kapsama miktarını artırmayı hedeflemiş ve hata lokalizasyonu yapmıştır. Ayrıca aracın geliştirme sürecine entegrasyonu gerçekleştirilmiştir. Burada kullanılacak örneklerin belirlenmesi ve kombinasyonel etkileşimin yapılandırılması zor bir problemdir.

Adaptif rastgele test, uygulaması kolay, basit ve popüler bir test üretim tekniğidir. Spesifikasyonlar eksikse veya kaynak koda erişim sağlanamıyor ise kullanımı oldukça yaygındır [14]. Adaptif test tekniği random test tekniğini iyileştirmek için yayınlanmıştır [36, 37]. Yapılan çalışmalara göre başarısız olan girdilerin belirli bir aralık boyunca da başarısız girdiler dizisi olduğunu göstermiştir [38]. Bu nedenle rastgele olarak denenen bir test verisinin başarısız olması durumunda başarılı test verisinin daha uzak bir bölgede olma ihtimali yüksektir. Tahmine dayalı bir yöntem olduğu için deterministik değildir. Bu alanda çeşitli algoritmalar geliştirilmiştir [36,37,39–43]. Bu çalışmalar arasında en iyi test verisinin bir sonraki test verisi üretimi için aday olarak seçilmesi, daha önce üretilmiş test verileri bölgelerinin hariç tutulması veya o bölgeleri parçalara bölüp uygun test verisinin seçilmesi gibi teknikler yer almaktadır [14]. C++ Test², AgitarOne³, Randoop [44],

¹<https://csrc.nist.gov/projects/automated-combinatorial-testing-for-software>

²<https://www.parasoft.com/product/cpptest>

³<http://www.agitar.com/solutions/products/agitarone.html>

Jwalk [45], GRT [46], T3 [47] ve JCrasher [48] gibi uygulamalar rastgele arama temelli sektördeki yazılımlardır.

Arama tabanlı test verisi/senaryosu üretimi arama tabanlı uygulama geliştirmenin alanlarından biridir [49]. Bu alanda ilk yayın Webb Miller ve David Spooner tarafından 1976 yılında gerçekleştirilmiştir [50]. Bu yayında, kayan noktalı sayıları giriş olarak alan programlar için bir çözüm üreten teknik geliştirilmiştir. Bilgisayarlarda gerçekleşen iyileşmeler ve meta-sezgisel algoritmaların kullanımının yaygınlaşması ile birlikte Korel'in ortaya koyduğu çalışmalar [51, 52] ve 1992 yılında Xanthakis'in genetik algoritma kullanarak çözüme ulaşması [53] ile arama tabanlı test verisi üretimi popüler hale gelmiştir. Bu alanda çok sayıda yayın olmasına karşın bunların çoğu yapısal metodoloji ile üretilmiş yazılımlar için geliştirilmiştir ve nesne yönelimli yazılımların meta-sezgisel yapay zeka algoritmaları ile test edilmesine yönelik yayınların sayısı oldukça azdır. Nesne yönelimli yazılımlar için arama tabanlı test senaryosu geliştiren uygulamalara EvoSuite [54], JTEExpert [55] örnek verilebilmektedir. RESTful web servisler için sistem seviyesinde test senaryosu üretiminde ise EvoMaster [56] bulunmaktadır.

Arama tabanlı test verisi üretimi ayrı bir problem olarak sınıflandırılmaktadır ve bu problemin çözümünde meta-sezgisel algoritmalar kullanılabilmektedir. Harman ve Jones [49] yazılım mühendisliği problemi olan test verisi üretiminde meta-sezgisel algoritmalar kullanımının, kapsama metriğine bağlı amaç fonksiyonu olarak tasarlanması ile verimli sonuçlar üretebileceğini öne sürmüştür. Meta-sezgisel algoritmalar, problemin karakteristiğinden bağımsız olarak amaç fonksiyonu elde edilebilen bütün problemlerde kullanılabilmektedir. Bu nedenle test verisi üretiminde de kullanımları oldukça yaygındır. Arama tabanlı test verisi üretimi hakkında yapılan çalışmalar, Harman ve Jones [49], Phil McMinn [57], Afzal ve ark. [58], Rähä [59], P. McMinn [60], Harman ve ark. (2009) [61], Harman ve ark. (2012) [62] tarafından derlenmiştir. Bütün bu derlemelerden anlaşıldığı üzere arama tabanlı yazılım mühendisliği alanı her geçen gün daha da popüler hale gelmektedir.

1.2. Arama Tabanlı Yazılım Testinin Temelleri

Bir yazılımda değişiklikler yapıldıktan sonra gerçekleştirilen bakım faaliyetleri sırasında, eklenen yeni bileşenlerin veya yazılımın değiştirilmiş kısımlarının test edilmesi (regresyon testi) gerekmektedir. Regresyon testi sonucunda, yazılımın hedeflendiği gibi davranışlar sergileyip sergilemediği belirlenebildiği için yazılım testi hem yazılım geliştirmede hem de bakım aşamasında oldukça önemlidir. Regresyon testi, değişikliklerin sistem davranışını değiştirmediği veya sistemin bütünlüğünü koruyup koruyamadığını doğrulamak için geçmişte yazılmış olan testleri yeniden çalıştırarak inceler. Bu işlem sırasında, bazı eski test senaryolarının güncellenmesi gerekmektedir. Ayrıca geliştirme sürecinde veya yazılım bakım süreçleri sırasında eklenen yeni bileşenlerin test edilmesi için yeni senaryolarının yazılması oldukça önemlidir. Genelde bu test senaryolarının yazılması da test uzmanları tarafından el ile gerçekleştirilmektedir. Test uzmanları özellikleri incelemeli, neyi test edeceklerini ve test senaryosu yazarken hangi stratejiyi kullanacaklarına karar vermelidir. Bu sebeple yazılım testinin yazılması oldukça maliyetli, zor ve hataya açık bir süreçtir.

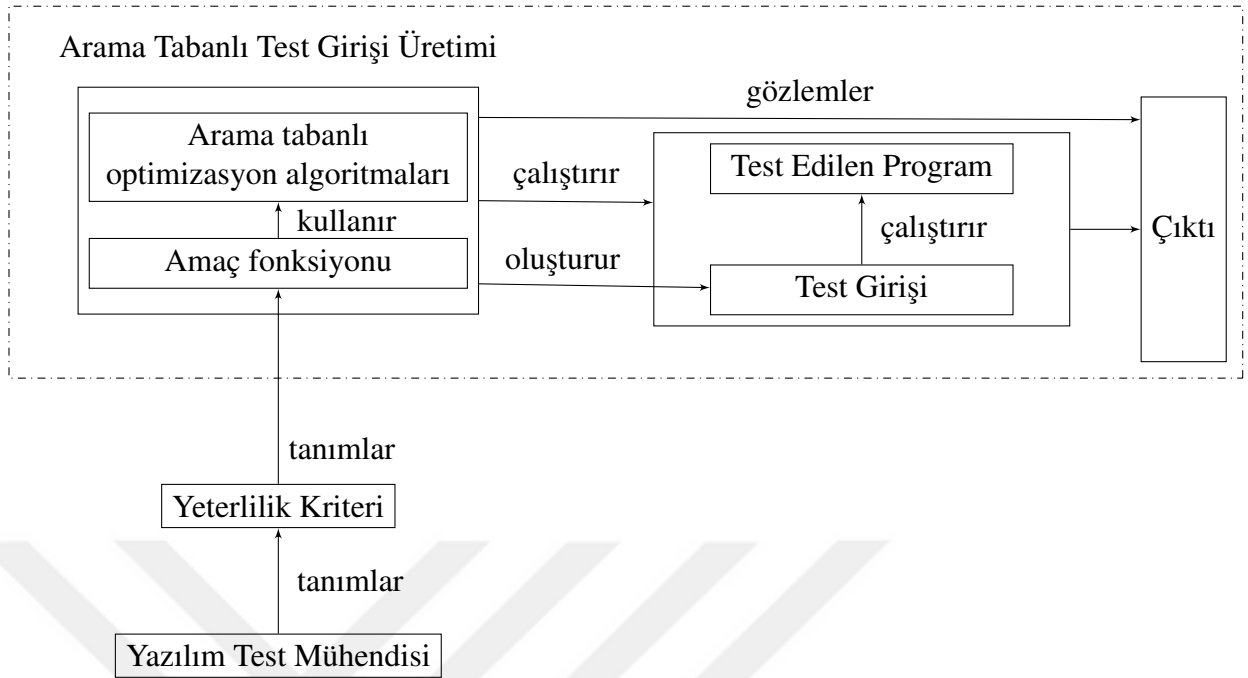
El ile test oluşturma'nın yüksek maliyetini düşürmek ve aynı zamanda test süreçlerinin güvenilirliğini artırmak için araştırmacılar bu süreci otomatik hale getirmeye çalışmıştır. Bu işi otomatik hale getirmenin en basit yolu ise bir yazılımın tüm olası girdileri ve ön koşul kombinasyonlarını göz önünde bulundurarak test etmekten ibarettir ancak bu oldukça maliyetlidir ve uygulanabilir bir yaklaşım değildir. Bu sebeple pek çok araştırmacı belirli kriterlere göre uygun testlerin oluşturulması problemini bir optimizasyon problemi olarak tanımlamıştır. Test kriterinin seçimi ise amaca ve otomatize edilecek testin tipine göre farklılık göstermektedir. Örneğin, çalışma zamanı gibi fonksiyonel olmayan özellikler test edilebilir veya güvenlik testi için hata durumlarının öncesi ve sonraki durumlarının testi gerçekleştirilebilmektedir. Otomatik yapısal test ele alındığında, çalıştırılan program yapısının elemanlarını en üst seviyeye çıkarabilmek amacıyla test senaryoları üretilebilmektedir. Bütün bu yaklaşımlar arama tabanlı yaklaşımdır ve SBST olarak genellenmektedir. Genel olarak literatürdeki çalışmaların büyük bir çoğunluğu SBST alanındadır [63–67].

Otomatik test üretim teknikleri, değiştirilmiş yazılımın, spesifikasyonları sağlamaya devam edip etmediğinin ve yeni değişikliklerin yazılımın değişmemiş kısımlarında sorunlara sebebiyet verip vermediğinin kontrol edilmesi için regresyon testinde de kullanılmaktadır [68]. Bu amaçla, test paketindeki mevcut bütün test senaryoları çalıştırılarak sistemin yeniden test edilmesi için otomatik bir araç kullanılabilir. Ardından değişikliklerden önce ve sonra elde edilen test sonuçları otomatik olarak karşılaştırılıp değişikliklerin neden olduğu kusurların otomatik olarak belirlenmesine yardımcı olabilmektedir. Bununla birlikte yazılım yaşam döngüsü boyunca test paketleri büyüme eğiliminde olduğundan bu strateji büyük sistemler için oldukça maliyetlidir ve uygulanabilir olmayabilir. Do ve ark. [69] 30,000'den fazla test senaryosu içeren bir test paketinin yürütülmesi için 1000 işlem saati gerektirdiğini söylemiştir. Bu tahmin sadece regresyon testinin çalışma süresidir ve test uzmanlarının bu süreci denetlemek, test çalışmalarını oluşturmak ve bu sonuçları izlemek için gerekli olan süreyi dikkate almamaktadır [63].

Regresyon testi için gerekli çabayı azaltmak için literatürde çeşitli stratejiler bulunmaktadır [68]. Test paketi minimizasyonu veya indirgemesi olarak adlandırılan yöntem test maliyetini düşürme yöntemlerinden biridir ve çeşitli kriterlere göre fazlalık olan test senaryoları test paketinden kaldırılarak paketin boyunun düşürülmesi amaçlanmıştır [70–78]. Bir diğer grup ise test senaryolarının bir alt kümesini seçmeyi amaçlayan yöntemlerdir [79–83]. Son olarak test senaryosu önceliklendirme yöntemleri ile hataları daha önce ortaya çıkaran test senaryolarının daha önce çalıştırılması amaçlanmıştır [80–85].

Arama tabanlı test verisi/senaryosu üretiminin temel şeması Şekil 1.1'de verilmiştir. Arama tabanlı optimizasyon algoritmaları doğrulayıcı (yazılım test mühendisi, yeterlilik kriterleri) doğrultusunda bir amaç fonksiyonu yardımıyla test girişlerini üretir, daha sonra sisteme giriş olarak verir. Sistem ise geri dönüşler üreterek amaç fonksiyonunu, amaç fonksiyonu ise optimizasyon algoritmasını yönlendirmektedir. Arama sonucunda bir test paketi son kullanıcıya çıktı olarak verilmektedir.

P bir program, I bütün olası girişleri temsil ederken, O bütün olası çıktıları temsil etmek üzere, $P : I \leftarrow O$ fonksiyonu ile ifade edilebilir. I içerisinde belirli bir veri olan x ile P



Şekil 1.1. Genel arama tabanlı yazılım test senaryosu/verisi üretim şeması

programının yürütülmesi ($P(x)$), P 'deki yapısal öğelerin (dallar ve komutlar) belirli bir çalıştırma sırasını ifade etmektedir. Farklı veri girişleri, P 'deki öğelerin farklı yürütme dizilerini türetebilmektedir. Bu sebeple, SBST yaklaşımının sezgisel amacı, P 'deki tüm öğeleri yürütmeye izin veren girdilerin (test senaryoları) oluşturulmasıdır. Bu nedenle P 'nin yapısı, test edilen elemanları analiz etmek için gereklidir [63].

Böyle bir analizi gerçekleştirmek için ise programın grafiksel gösterimi olan kontrol akış çizgesi (control flow graph, (CFG)) kullanılmaktadır. CFG program akışının grafiksel gösterimidir. CFG yönlendirilmiş çizge olup bir adet giriş ve bir adet de çıkışa sahiptir. Giriş düğümü programın başlangıç noktasını, çıkış düğümü ise programın sonlandığı noktayı belirtmektedir [57]. CFG üzerindeki düğümler komutları (statement), kenarlar (edge) ise akış kontrolünü ifade etmektedir. Bir CFG, N düğümler kümesi (statement nodes), E , düğümler ve etiketler arasındaki geçişler, s programın başlayacağı tekil başlangıç düğümü, e ise programın sonlanacağı düğüm olmak üzere $G = (N, E, s, e)$ olarak tanımlanan bir yönlendirilmiş çizgedir. Gerçekleştirilmiş pek çok çalışmada CFG önemli bir yere sahiptir.

Karar komutlarına karşılık gelen düğümler (if, while gibi) *dallanma düğümleri* (branching nodes) olarak adlandırılmaktadır. Bu düğümlerden çıkan kenarlara ise *dal* (branch) adı

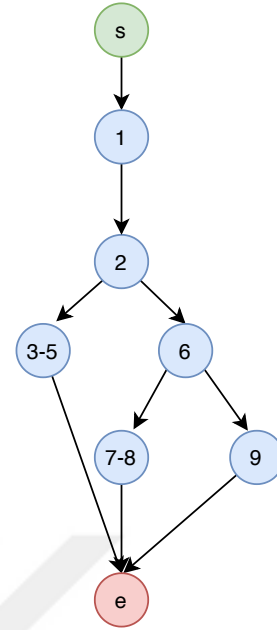
verilir. Bir dalın geçilip geçilemeyeceğini belirleyen koşula ise *dal karşılaştırma belirtimi* (branch predicate) adı verilmektedir. Belirli bir kenarı geçmek için karşılık gelen kenarın koşulunu sağlaması gerekmektedir.

Örnek bir kontrol akış çizgesi ve buna ait kod parçası Şekil 1.2’de verilmiştir. Her bir düğüm programdaki komutlara denk gelmektedir ve üzerinde yazan sayılar satır numarasıdır. Bu program ikinci dereceden bir denklemin üç parametresini almaktadır ve verilen denklemin kökü varsa bu kökleri bulmakta, yoksa kökü bulunmadığını söylemektedir. Çakışık kök olması durumunda ($b^2 - 4ac == 0$) programın $\{s, 1, 2, 6, 7, 8, e\}$ düğümlerinden oluşan dal çalışmaktadır ve diğer düğümlerinin çalışması olanaksızdır.

Bir p yolu, $p = \{n_1, n_2, \dots, n_m\}$ düğüm serilerinden oluşur ve her bir ardışık düğüm çifti (n_i, n_{i+1}) , n_i ’den n_{i+1} ’e doğru yönlendirilmiş kenar ile birbirine bağlıdır ($\forall i, 1 \leq i < n, (n_i, n_{i+1} \in E)$). Her $P(x)$ çalıştırılmasında p ’deki bütün kenarlar çalışır ve x testinin p yolunu kapsadığı söylenir. Eğer bir yol erişilebilirse (feasible) o yolu kapsayacak bir girdi bulunmaktadır. Aksi takdirde o yolun erişilemez (infeasible) olduğu söylenir. Eğer bir yol e başlangıç düğümü ile başlar ve e çıkış düğümü ile biterse *tamamlanmış yol* (complete path) adı verilir. Aksi takdirde *tamamlanmamış yol* (incomplete path) veya *yol parçası* (path segment) adı verilmektedir. Örneğin Şekil 1.2’de bulunan $\{s, 1, 2, 6, 7, 8, e\}$ yolu erişilebilir tamamlanmış bir yoldur ve $a = 2, b = 4, c = 2$ verileri ile kapsanabilmektedir.

CFG içinde program değişkenlerini değiştiren düğümler (atama veya girdi komutları) *tanım düğümü* (definition node) olarak adlandırılmaktadır. Örneğin Şekil 1.2’de 1. düğüm *result* değişkeni için tanım düğümüdür. CFG içinde bir değişkenin kullanıldığı atama komutları, çıktı komutları veya dallanma şartlarındaki düğümlere *kullanım düğümü* (use node) denmektedir. Örneğin 2. düğüm *result* değişkeni için kullanım düğümüdür. Program değişkenini değiştirmeyen düğümlere ise *tanım değiştirmeyen yol* (definition-clear path) adı verilmektedir. Örneğin $p = \{s, 1, 2, 6, 9\}$ $r1$ ve $r2$ için tanım değiştirmeyen yoldur.

Düğüm	Komut
s	void denklem_kokleri(double a, double b, double c)
1	{
2	double result = b * b - 4.0 * a * c;
3-5	if (result > 0.0) { double r1 = (-b + Math.pow(result, 0.5)) / (2.0 * a); double r2 = (-b - Math.pow(result, 0.5)) / (2.0 * a); println("Kökler:" + r1 + " ve " + r2); }
6	else {
7-8	if (result == 0.0) { double r1 = -b / (2.0 * a); println("Kök: " + r1); }
9	else { println("Reel kök bulunmamaktadır."); }
e	return; }



Şekil 1.2. İkinci dereceden denklem kökleri bulan kod parçası ve CFG gösterimi.

CFG, programın yürütülmesi sırasında çalıştırılan öğelerin takibi için kullanılmaktadır. Bu nedenle CFG ve dinamik analiz programın hangi bölümlerinin test edildiğini anlamak için kullanılabilir. Bu da kapsaması/test edilmesi gereken kod parçalarının türüne göre birçok test kapsam ölçütünün tanımlanmasını sağlamıştır. En çok kullanılan test kapsam kriterleri aşağıda verilmiştir [86, 87]:

Komut Kapsamı (statement coverage): CFG içerisindeki tüm komutların çalıştırılmasını hedeflemektedir.

Dal Kapsamı (branch coverage): CFG üzerindeki tüm dalların çalıştırılmasını hedeflemektedir. Bir if komutunun hem true, hem de false yolunun çalışmış olması gerekmektedir.

Şart Kapsamı (condition coverage): Her bir şarttaki true ve false durumlarının kapsanmasını hedeflemektedir.

Çoklu Şart Kapsamı (multiple-condition coverage): Her bir şart içerisindeki alt şartların kombinasyonlarının değerlerinin kapsanmasını amaçlamaktadır. Örneği if(A||B) durumu için (true,true), (true,false), (false,true) ve (false,false) olmak üzere toplam 4 farklı durum bulunmaktadır.

Yol Kapsamı (Path Coverage): CFG üzerindeki tüm yolların kapsanmasını hedeflemektedir.

Komut kapsamı, bu kapsamlar içerisinde en yetersiz olanıdır çünkü CFG üzerindeki belirli yollarda ortaya çıkabilecek hataları belirleyememektedir. Diğer kriterler daha işlevsel olsalar da yol veya şart kapsamı gibi kapsamlar büyük veya karmaşık programlar için uygun olmadığından dal kapsamı yazılım testinde en çok kullanılan kriterdir [57].

Kapsam kriterleri, otomatik test verisi/senaryosu oluşturma problemi formüle ederken ele alınabilecek en uygun sezgisel amaç fonksiyonlarından biridir. Bununla birlikte, basitliğine rağmen bu yaklaşım genellikle şans eseri kapsanabilir olan yapılara rehberlik sağlayamamaktadır. Büyük bir arama uzayı ele alındığında, yalnızca özel bir sayının giriş olduğu durumlarda true değerinin döndüğü karşılaştırma belirtileri bu duruma bir örnektir. Örneğin, $if(a == k)$ komutunu ele aldığımızda, k bir sabittir ve yalnızca a değeri girdi olduğunda bu karşılaştırma belirtimi true olacaktır. Bunun dışında bütün durumlarda false değerini alacaktır ve bu duruma Bayrak Problemi (flag problem) adı verilmektedir. Eğer dal kapsamı metriği amaç fonksiyonu olarak kullanılırsa, amaç fonksiyonu değerinin, düşük amaç fonksiyonu değerinden, yüksek amaç fonksiyonu değerine doğru herhangi bir arama yönünü belirleyecek rolü olamayacaktır. Hatta $if(a >= k)$ gibi durumlarda dahi amaç fonksiyonu aramaya rehberlik edemeyecektir [63].

Bu problemin üstesinden gelebilmek için önceki çalışmalar [50, 51, 53, 63] uygunluk fonksiyonunu yönlendirilebilir olarak ve kapsama miktarını da durdurma kriteri olarak (maksimum kapsama miktarına erişene kadar veya belli bir zaman iyileştirme olmayana kadar) kullanılmasını önermiştir. Fonksiyonun yönlendirilmesi için en sık kullanılan yöntem dal mesafesi (branch distance) olup Bölüm 1.2.2’de söz edilmektedir.

1.2.1. Temsil Biçimi

SBST’de problemin arama algoritmaları ile çözülebilmesi için öncelikle gösteriminin belirlenmesi gerekmektedir. Bir test verisi aday çözüm kümesi, P programının giriş değişkenleri için bir girdi değerler kümesi olarak temsil edilen test senaryosudur. $I =$

$(x_1, x_2, \dots, x_k)$, P programının girdi parametreleridir. D_{x_i} alanı ise x_i girdisinin etki alanını temsil etmektedir. P programının alanı olan D_p ise çapraz çarpımlardan oluşur ve $D_p = D_{x_1} \times D_{x_2} \times \dots \times D_{x_k}$ olarak tanımlanmaktadır. Bu tanım da arama algoritmaları için gösterim olarak kullanılabilir [63].

Birim test senaryosunda bir test senaryosu t , yalnızca I girdi vektörlerinden değil bu girdi vektörleri ile birlikte P programını yürüten ve hedef programlama dilinde oluşturulmuş bir programdır. Bu nedenle bir test oluşturma aracı, bir girdi vektörünü ve bu girdi değişkenlerini kullanan komutları bir arada oluşturmalıdır. Bir optimizasyon algoritması doğal olarak bu programları, komutları, nesneleri anlayamaz ve bu temsile izin veren bir programın kullanılması gerekmektedir [88].

Fraser ve Arcuri (2011) [89] ve Fraser ve Arcuri (2012) [90] çalışmalarına göre, bir test senaryosu sıralı komutlar bütünüdür ($t = \{s_1, s_2, \dots, s_l\}$). Her bir komut, hedef nesne, metot ve parametre/değişkenler'den oluşmaktadır. Nesne yönelimli programlama dilleri için komutlar ile ilgili ayrıntılı bilgi Bölüm 3.5.1'de verilmiştir.

1.2.2. Amaç Fonksiyonları

SBST yaklaşımları, üretilen çözümlerin kalitesini ölçen ve aramayı muhtemel iyi bölgelere yönlendiren uygun bir amaç fonksiyonunun tasarımına ihtiyaç duymaktadır. Bu nedenle araştırmacılar, yapısal kapsam (structural coverage) [86], yaklaşım seviyesi (approach level) [91], mesafe hesaplama (distance calculation) [90, 92] veya birden fazla ölçümün kombinasyonu [93, 94] gibi farklı teknikleri kullanan amaç fonksiyon tanımları önermişlerdir. Bununla birlikte, hangi yaklaşımın hangi koşullarda daha iyi sonuç vereceği bilinmemektedir [95].

Test senaryosu üretiminde bir tekniği başarılı bir şekilde uyarlayabilmek için amaç fonksiyonunun, problemin gösteriminin ve yeni çözüm üretim mekanizmalarının tanımlanması gerekmektedir [95, 96].

Yapısal test tekniğinde yeterlilik kriteri, sayısal olarak ölçülüp değerlendirilebilen, kapsanan dal veya komutlar gibi bir test hedefidir. Bu kriter, aday test girdilerinin performansını değerlendirmek için kullanılmaktadır. Program, aday çözümlerin

uygunluğunu değerlendirmek için üretilen girdilerle çalıştırılır. Amaç fonksiyonu, arama eyleminde oldukça önemli bir yere sahiptir ve iyi tanımlanmış bir amaç fonksiyonu çözüm bulma ve daha yüksek kapsama miktarına erişme olasılığını artırır. Ayrıca arama sürecinin daha iyi yönlendirilmesi, tüketilen sistem kaynaklarının azalmasına yardımcı olur [65,95,97].

Temel olarak kullanmış olduğu bilgiye göre farklılık gösteren, dal mesafesi tabanlı (branch-distance-oriented), kontrol tabanlı (control-oriented) ve karma yaklaşım (combined approaches)'dan oluşan üç farklı amaç fonksiyonu türü vardır. Genel olarak amaç fonksiyonları minimize edilecek bir f fonksiyonu olarak tanımlanmaktadır ($f(t) = 0$) ve yalnızca t testi çalıştığında hedef dal kapsanmış olur.

1.2.2.1. Dal Mesafesi Tabanlı Yaklaşım

Dal mesafesi tabanlı amaç fonksiyonları (f), programın karşılaştırma belirtileri (predicate) dikkate alınarak tanımlanmaktadır ve verilen bir girdi kullanılarak, yürütülen akış çizgesindeki hedef yolu veya dalı geçmeye ne kadar uzaklıkta olduğunu ölçmektedir. Örneğin $a==b$ şartı ile kapsanan dal, $abs(a - b)$ ifadesi ile bir amaç fonksiyonuna dönüştürülür. Bu fonksiyon, a ve b 'nin ilgili dalı kapsamaya ne kadar uzaklıkta olduğunu gösterir. Dal mesafesi değeri ne kadar düşükse o çözüme o kadar yaklaşılmış demektir.

Tracey ve ark. [98,99], Korel [51] tarafından tanımlanan fonksiyonlardan esinlenerek dal mesafesi tabanlı bir amaç fonksiyonu önermiştir (Tablo 1.1). CFG'deki her koşul için dal mesafesi f , girdi olarak I ile ifade edilen bir dizi değer alır ve I 'nın mevcut değerleri ile sayısal ifadelerle çevrilip hesaplamalar gerçekleştirilmektedir. Bu f fonksiyonu integer, float, double gibi sayısal ifadelerle mantıksal ifadeler için gayet iyi çalışmaktadır. Daha karmaşık tipler olan nesneler, string ifadeler gibi yapılar için ise literatürde daha farklı yaklaşımlar bulunmaktadır [60,65,100]. Hesaplamanın en sonunda ise bütün dallar veya yollar için bütün dal mesafe değerleri toplanarak amaç fonksiyonu olarak sunulmaktadır.

Dal mesafesi amaç fonksiyonu olarak kullanıldığında, arama algoritması CFG üzerindeki belirtilen yol kapsanana kadar çalışmaktadır. Arama süreci ise CFG üzerindeki bütün elemanlar kapsanana kadar devam eder. Xanthakis ve ark. [53] Genetik Algoritmayı (GA), CFG üzerindeki yolları kapsamak için test verisi üretiminde kullanmıştır ancak

bu çalışmada, test edilecek yolları test uzmanı belirlemektedir. Jones ve ark. [101] ise yol kapsamı kriterini de dikkate alarak yol seçim problemini ortadan kaldırmış ve amaç fonksiyonunu buna göre tanımlamıştır.

Dal mesafesi yaklaşımının en önemli problemlerinden biri boole değeri içeren koşullardaki bayrak problemidir. Bu tür koşullar true ve false olmak üzere iki değer aldığı için arama algoritmasına rehberlik edememektedir [63]. Pek çok araştırmacı bu problemi çözebilmek için test edilebilirlik dönüşümleri [64,67] ve daha kapsamlı amaç fonksiyonları önermişlerdir [66].

Tablo 1.1. Tracey'nin önermiş olduğu dal mesafesi fonksiyonu.

Karşılaştırma Belirtimi	Amaç Fonksiyonu
Boolean	if <i>TRUE</i> then 0 else <i>K</i>
$a = b$	if $abs(a - b) = 0$ then 0 else $abs(a - b) + K$
$a \neq b$	if $abs(a - b) \neq 0$ then 0 else $abs(a - b) + K$
$a < b$	if $a < b$ then 0 else $a - b + K$
$a \leq b$	if $a \leq b$ then 0 else $a - b + K$
$a > b$	if $a > b$ then 0 else $b - a + K$
$a \geq b$	if $a \geq b$ then 0 else $b - a + K$

1.2.2.2. Kontrol Tabanlı Yaklaşımlar

Kontrol tabanlı yaklaşımda amaç fonksiyonu, çalıştırılması gereken düğümler ile tanımlanmaktadır. Jones ve ark. [101] var olan ve çalıştırılan komut sayısı arasındaki farkı amaç fonksiyonu olarak kullanmıştır. Pargas ve ark. [91] bir programı, belirli bir girdi ile çalıştırırken yürütülen program öğelerinin sayısını ölçen bir amaç fonksiyonu tanımlamıştır. Bu çalışmada, CFG'in yapısını analiz ederek, dal kapsamı metriği kullanılmıştır. Yani *P* programına *I* girdisi verildiğinde, *bağımlı*, çalıştırılması gereken bağımlı düğüm sayısı olsun ve *çalıştırılan* da *P* programına *I* girdisi verildiğinde çalıştırılan düğüm sayısı olsun. Bu durumda Pargas ve ark.'ın [91] önermiş olduğu amaç fonksiyonu Eşitlik 1.1 ile tanımlanabilmektedir.

$$min f(I) = (\text{bağımlı} - \text{çalıştırılan}) \quad (1.1)$$

Kontrol tabanlı amaç fonksiyonunun dezavantajı ise arama sürecine rehberlik edememesidir. Bu fonksiyon karşılaştırma belirtimini dikkate almadığından, CFG'de aynı

bağımlı düğümleri çalıştıran tüm girdiler aynı uygunluk değerine sahiptir [63].

1.2.2.3. Karma Yaklaşımlar

Önceki yaklaşımlarda bulunan sınırlamaların üstesinden gelebilmek için Wegener ve ark. [94] dal mesafesini ve kontrol tabanlı yaklaşımı tek bir amaç fonksiyonunda birleştiren bir yaklaşım önermiştir. P programına I girdisi verildiğinde, *bağımlı*, çalıştırılması gereken bağımlı düğüm sayısı olsun ve *çalıştırılan* da P programına I girdisi verildiğinde çalıştırılan düğüm sayısı olsun. Wegener ve ark. [94] önermiş olduğu amaç fonksiyonu Eşitlik 1.2 ile hesaplanabilir.

$$\min f(I) = (\text{bağımlı} - \text{çalıştırılan} - 1) + \text{dal_mesafesi} \quad (1.2)$$

Bu denklemde (*bağımlı* - *çalıştırılan* - 1) ifadesi yaklaşım seviyesini (approach level) ifade etmektedir ve I girdisiyle elde edilen çalıştırılmamış bağımlı düğüm sayısını vermektedir. Eşitlikteki *dal_mesafesi* ifadesi ise dal mesafesi değerini vermektedir ve $[0, 1]$ aralığındadır. Dal mesafesi bir sonraki yaklaşım seviyesine ulaşabilmek için gereken I girdisinin ne kadar yaklaştığını belirtmektedir. Yaklaşım seviyesi ve dal mesafesinin birlikte kullanıldığı bu yaklaşım literatürde en sık kullanılan yöntemdir ve bu yaklaşımla ilgili pek çok çalışma gerçekleştirilmiştir [64, 90, 94, 102, 103].

1.3. Arama Tabanlı Yazılım Testi Hakkında Yapılan Çalışmalar

Harman ve Jones [49], ayrık problem türü olarak sınıflandırılan SBST'nin, meta-sezgisel algoritmalar kullanımının kapsama metriğine bağlı bir amaç fonksiyonu tasarımıyla mümkün olabileceğini öne sürmüştür. SBST hakkında yapılan çalışmalar; Harman ve Jones (2001) [49], McMinn (2004) [57], Afzal ve ark. (2009) [58], Răihä (2010) [59], McMinn (2011) [60], Harman ve ark. (2012a) [104], Harman ve ark. (2012b) [105], Anand ve ark. (2013) [14] ve Khari ve Kumar (2013) [106] tarafından derlenmiştir.

Son yıllarda çeşitli meta-sezgisellerin kullanıldığı, SBST üzerine çeşitli çalışmalar yapılmıştır. Windisch ve ark. [107] dal kapsamı (branch coverage) kriterini amaç fonksiyonu olarak kullanarak Parçacık Sürü Optimizasyonu (PSO) algoritması ve Genetik

Algoritmayı (GA) kıyaslamış ve PSO algoritmasının GA'ya göre daha iyi sonuçlar elde ettiğini göstermiştir. Tiwari ve ark. [108] PSO algoritmasını regresyon testi için uygun hale getirmiş ve beş benchmark üzerinde test etmiştir. Önermiş olduğu algoritma diğer PSO türlerinden daha başarılı olmuştur. Zhu ve ark. [109] PSO algoritmasına ait eylemsizlik ağırlığı (w) parametresinin dinamik olarak değiştiği bir PSO varyantı önermiştir ve bu önermiş olduğu modeli GA ve PSO algoritmaları ile kıyaslamıştır. Önermiş olduğu model bu iki algoritmadan daha başarılı olmuştur. Dahiya ve ark. [110] bütün yolların testi için hibrit bir PSO modeli önermiştir. Yapısal test problemleri üzerinde gerçekleştirilen deneylere göre, önerilen algoritmanın daha gürbüz olduğu belirtilmiştir. Singla ve ark. [111] yazılım testi için GA ve PSO algoritmalarının birleşiminden oluşan hibrit bir model önermiştir. Önerilen bu yöntemi GA ve PSO algoritmaları ile kıyaslamış ve önermiş olduğu algoritmanın bu algoritmalarından daha iyi sonuç elde ettiğini bulmuştur. Latiu ve ark. [112] yakınsama seviyesi + dal mesafesi amaç fonksiyonlarını kullanarak GA, PSO ve Işıl İşlem (Simulated Annealing, SA) algoritmalarını test etmiş ve bu algoritmaların etkin olarak kullanılabileceğini göstermiştir. Landa Becerra ve ark. [113], dal kapsamı metriğini amaç fonksiyonu olarak kullanarak temel Evrimsel Gelişim (DE) ve Breeder DE algoritmalarını kıyaslamış ve DE algoritmalarının gerçek dünya problemlerinin çözümü için uygun olduğunu söylemiştir. Jianfeng ve ark. [114], DE tabanlı bir test verisi üretim algoritması geliştirmişler ve bu algoritmayı SA, GA, karınca koloni (ACO), Cross-Entropy ve PSO algoritmaları ile kıyaslayarak önermiş oldukları algoritmanın daha iyi sonuç verdiğini göstermişlerdir. Liang ve ark. [115], DE tabanlı bir test senaryosu üretim algoritması önermiş ve farklı mutasyon operatörleri ile kontrol parametre değerlerini analiz etmişlerdir. Çalışmadan elde edilen sonuca göre önermiş oldukları yaklaşımın iyi sonuçlar ürettiği söylenmiştir. Mala ve ark. ABC algoritmasını test verisi üretiminde kullanmıştır. Yapmış oldukları ilk çalışmada [116] ABC ve GA algoritmalarını, ikinci çalışmada [117] ise ABC ve ACO algoritmalarını kıyaslamıştır. Bu çalışmalarda kapsama tabanlı amaç fonksiyonu kullanılmış olup, önerilen yöntemin daha az hesaplama zamanına ihtiyaç duyduğu ve daha ölçeklenebilir olduğu ifade edilmiştir. Yapmış oldukları bir başka çalışmada ise [118] ABC algoritmasını paralel hale getirerek hesaplama zamanını düşürmüşler ve bu algoritmayı ABC, GA ve RS algoritmaları ile kıyaslamışlardır. Önerilen yaklaşım diğer yaklaşımlardan daha hızlı sonuca ulaşmıştır. Dahiya ve ark. [119] dal kapsamı temelli

bir amaç fonksiyonu kullanarak bir ABC modeli önermiştir. Önerilen bu model 10 gerçek dünya problemi üzerinde çalıştırılmış ve önerilen modelin zor problemlerde iyi sonuçlar elde ettiği görülmüştür. Lam ve ark. [120] daha kısa çalışma zamanında sonuç üretmeye dayalı paralel bir ABC modeli önermiştir. Elde edilen sonuçlara göre önerilen model bölgesel minimumlara takılmadan başarılı sonuçlar üretmiştir. Malhotra and Khari [121] GA algoritmasındaki mutasyon operatörünü kullanarak ABC algoritması varyantı geliştirmiş ve 10 adet C++ programı üzerinde çalıştırmıştır. Elde edilen sonuçlara göre önerilen yöntem daha başarılı olmuştur. Malhotra ve ark. [122] yol kapsama metriğini amaç fonksiyonu olarak kullanarak ABC, ACO ve GA algoritmalarını 9 C++ program üzerinde çalıştırmıştır. Bu çalışmaya göre, ABC algoritması diğer algoritmalarından daha başarılı olmuştur. Suri ve Kaur [123] regresyon testi için ABC algoritmasını kullanmıştır. Çalışma 8 örnek üzerinde yapılmış ve %100 kapsama miktarına erişilmiştir. Sahin ve Akay [103], ABC, PSO, DE, Ateş Böceği (FA) ve RS algoritmalarının performanslarını çeşitli amaç fonksiyonları ile analiz etmiştir. Elde ettiği sonuçlara göre yakınsama seviyesi + dal mesafesi tabanlı uygunluk fonksiyonu ile en yüksek kapsama miktarına erişilmiştir. Srivatsava ve ark. [124] FA temelli bir model önermiştir. Matris tabanlı bir amaç fonksiyonu kullanarak indirgeme yapan bu yöntem ile optimum sayıda test yolları üretilmiştir.

2. BÖLÜM

LİTERATÜR TARAMASI

Test senaryosu üretimi üzerine yapılan çalışmalar 1970’lerde başlamıştır. Zaman içerisinde bilgisayarlardaki iyileşmeler ve optimizasyon algoritmaların gelişmesi ile birlikte SBST alanına daha fazla yoğunlaşmış ve çeşitli yaklaşımlar önerilmiştir. Bu yöntemleri oldukça kapsamlı bir şekilde inceleyen derlemeler yayınlanmıştır [7, 14, 57–60, 104–106].

Rastgele Arama (Random Search, (RS)), önerilen ilk tekniklerden biridir ve en temel test tekniğidir [14]. Bu yöntemin uygulanması oldukça kolaydır ve diğer yöntemlerle de kolaylıkla entegre olabilmektedir. Sistemin spesifikasyonları tamamlanmasa veya kaynak kodları erişilebilir olmasa dahi RS uygulanabilir bir teknik olmaya devam etmektedir [14].

Adaptif Rastgele Arama, test senaryolarının başarısız olduğu noktaların yakınlarında da başarısız olabileceği varsayımına dayanmaktadır ve RS algoritmasının geliştirilmiş bir versiyonudur [36, 37, 125]. Bununla birlikte bu yöntemin oldukça basit problemlerde bile başarısız olduğu bulunmuştur. Örneğin test verisi üretiminin benchmark problemlerinden olan üçgen sınıflandırma probleminde RS oldukça kısa süre içerisinde sonuç elde ederken, Adaptif Rastgele Arama oldukça uzun sürede sonuca erişebilmiştir [126]. Bu nedenle literatürde RS çok daha fazla kullanılmaktadır [95].

SBST için genetik algoritma [96, 127–129] gibi optimizasyon algoritmaları kullanılmaktadır. Bu algoritmalar bir amaç fonksiyonu yardımıyla yüksek kaliteli çözümler üretmeye çalışmaktadır. SBST, yapısal [102, 103, 130–132], fonksiyonel [133], fonksiyonel olmayan [134] ve duruma dayalı özellikler [135] dahil olmak üzere çok çeşitli test hedeflerine uygulanmıştır. Entegrasyon testi [136, 137], mutasyon testi [138, 139], regresyon testi [68, 82, 140], stres testi [141] ve web uygulamaları [142, 143] gibi

yaklaşımları test etmek için farklı SBST teknikleri geliştirilmiştir.

Bu tez çalışmasında; test senaryosu üretimi, nesne yönelimli programlama için yazılım test senaryosu üretimi ve RESTful web servisler için test senaryosu üretimi olmak üzere iki alt başlıkta incelenmiştir.

2.1. Nesne Yönelimli Programlama için Test Senaryosu Üretiminde Yapılan Çalışmalar

Son yıllarda bazı araştırmacılar nesne yönelimli programlama için arama tabanlı test verisi üretimi yöntemleri üzerine çalışma yürütmüşlerdir [54, 88, 102, 144–152].

Tonella [102], nesne yönelimli programlama için test üretiminde SBST tekniğini ilk uygulayan araştırmacı olmuştur. Yapılan çalışmada kapsama değerini maksimum ve test kümesini minimum yapmak için GA tabanlı test senaryo üretim tekniği kullanılmıştır ve bu teknik 6 farklı sınıfa uygulanmıştır. Çalışmada, evrimsel bir yaklaşımın test üretiminde başarılı sonuçlar elde edebileceği gösterilmiştir. Wappler ve Lammermann [88] evrimsel algoritmalar ile nesne yönelimli programlar için birim testler üretmiştir. Üretilen test senaryoları ile satır, dal ve şart kapsama metriklerini maksimize etmeye çalışmıştır. Wappler ve Wegener [144, 145] test üretimi için Strongly-Typed Genetik Programlama (STGP) kullanmıştır. Bu çalışmada testlerin çağırılma sırasının iyileştirilmesi sağlanmıştır. Liaskos ve ark. [146] veri akış (data-flow) kapsama metriğini kullanarak nesne yönelimli programlama için test üretimi gerçekleştirilmiştir. Elde ettiği sonuçlara göre veri akış kapsama metriği daha iyi sonuçlar verirken hesaplama zamanını artırmaktadır. Sagarna ve ark. [147] Dağıtım Algoritmalarının Tahmini (Estimation of Distribution Algorithms, EDAs) kullanarak belirli tipteki bir nesne için test verisi üretimi gerçekleştirilmiştir. Bu çalışmada kapları (container) üzerinde çalışılmıştır. Elde edilen sonuçlara göre EDAs algoritması test verisi üretiminde başarılı olmuştur. Wappler ve Schieferdecker [148] erişilebilir olmayan metotların otomatik olarak test üretimini iyileştirici çalışmalarda bulunmuştur. Aynı zamanda testler için performansı artırmak adına yeni bir gösterim önermişlerdir. Fraser ve Zeller [149] otomatik olarak birim testler üreten ve bunlardaki mutasyonları tespit eden bir yöntem önermişlerdir. Önerilen yöntem 10 açık kaynak kütüphane üzerinde denenmiş ve yöntemin başarılı olduğu gösterilmiştir. Fraser ve Arcuri [54] yeni bir yöntem deneyerek bütün kapsama

metriklerini aynı anda maksimum yapmaya çalışmıştır. Önerilen yöntem 1741 sınıf üzerinde denenmiştir. Önerilen yöntemin 188 kat daha iyi olduğu ve üretilen test senaryolarının da %62'ye kadar daha küçük olduğu görülmüştür. Fraser ve Arcuri [150] iki yeni yaklaşım önererek gereksiz test çalıştırılmalarından kaçınmış ve tam test üretimi yöntemini kullanmıştır. Rastgele seçilmiş 100 açık kaynak projedeki 8963 sınıf üzerinde çalıştırmıştır. Önerilen yöntemin başarılı olduğu görülmüştür. Campos ve ark. [151] önemli açık kaynak sınıflar üzerinde altı farklı algoritma çalıştırmışlardır. Elde edilen sonuçlara göre evrimsel algoritma, RS algoritmasına göre daha iyi sonuç elde etmiştir ve çoklu kriter daha efektif sonuçlar vermektedir. Rojas ve ark. [152] tek kriter ve tam test senaryo üretimi tekniklerini kıyaslamıştır. Çalışma 100 Java sınıfı üzerinde üç farklı kriter ile gerçekleştirilmiştir. Elde edilen sonuçlara göre tek bir kritere göre tam test senaryo üretimi daha iyi sonuçlar elde etmiştir. Aynı zamanda arşiv tekniği de genel olarak iyi sonuçlar üretse de bazı dezavantajları bulunmaktadır.

Nesne yönelimli programlama için arama tabanlı test senaryosu üretiminde çeşitli araçlar geliştirilmiştir. EvoSuite isimli araç Fraser ve Arcuri [54] tarafından, JTEExpert, Sakti ve ark. [55] tarafından, EvoPSO Shahabi [153] ve EvoTLBO da Mehdi ve ark. [154] tarafından geliştirilmiştir. EvoPSO ve EvoTLBO, EvoSuite temelli araçlardır. EvoTLBO Öğretme-öğrenme esaslı optimizasyon algoritması (Teaching-learning-based optimization, TLBO) yazılım test senaryosu üretimi için uygun hale getirilmiş, Standart Genetik Algoritma (SGA) ve Monotonik Genetik Algoritma (MGA) ile kıyaslanmıştır. EvoPSO ise PSO temelli olarak oluşturulmuş, SGA ve MGA algoritmaları ile kıyaslanmıştır. Her iki çalışmada da istatistiksel olarak fark elde edilememiş ve sonuçlar ayrıntılı olarak incelenmemiştir.

2.2. RESTful Servisler için Test Senaryosu Üretiminde Yapılan Çalışmalar

RESTful servisler için test senaryosu üretimine yapılan çalışmalar oldukça kısıtlıdır. Segura ve ark. [155] RESTful web servisleri test ederken, doğrulayıcı belirlemede metamorfik ilişkilerin kullanılabileceğini önermiştir. Çalışmada Youtube ve Spotify web servislerinin 60 metamorfik ilişkisi belirlenmiştir. Web servislerin testinde meta-sezgisel tekniklerin kullanılması da son yıllarda popüler hale gelmektedir. Penta [156] Hizmet Seviyesi Anlaşması (SLA) ihlallerine neden olan girdiler ve yapılandırmalar oluşturmak

için GA tabanlı yaklaşım önermiştir. Önermiş olduğu yaklaşım ile ses işleme ve grafik oluşturma hizmetlerine müdahalede bulunmuş ve her iki durumda da, Hizmet Kalitesi kısıtlamalarını ihlal edebilen test verilerinin üretilebildiğini göstermiştir. Arcuri'nin RESTful web servis testi için pek çok çalışması bulunmaktadır. Arcuri [157] test senaryolarının meta-sezgisel bir algoritma ile otomatik olarak oluşturulduğu saydam kutu test yaklaşımı önermiştir. Çalışmalarını EvoMaster¹ isimli açık kaynak kodlu bir proje olarak paylaşmıştır ve projenin detaylarını [56] çalışmasında paylaşmıştır. Önermiş oldukları yöntem ile gerçek dünya problemleri üzerinde 38 hata bulmuştur fakat üretilen test senaryolarının kapsama miktarı el ile yazılan testlerden daha düşüktür. Arcuri [158] çalışmasında MIO algoritmasını önermiş ve performansını incelemiştir. Yapmış olduğu deney sonuçlarına göre WTS ve MOSA algoritmalarından daha iyi sonuç elde etmiştir. 2019 yılında yapılan bir başka çalışmada [143] ise; önermiş olduğu yöntemi, 22 bin satırdan daha fazla koda sahip 5 farklı projeyi test etmek için kullanmıştır ve yapmış olduğu testler sonucunda 80 farklı hata yakalamıştır. Bu çalışmada daha yüksek kapsama miktarına erişmeyi engelleyen şeyin SQL veri tabanı olduğu ve üzerine daha fazla çalışmanın gerektiğini vurgulamıştır. Arcuri ve Galeotti [159] SQL veri tabanını da ele alarak yeni bir arama tabanlı yaklaşım sunmuştur. Bu yaklaşım sayesinde %18'e kadar daha fazla kapsama miktarı elde etmiştir. Zhang ve ark. [160] HTTP kaynaklarının üretilmesinde yeni bir model ortaya koymuş ve MIO algoritmasında %40'a kadar iyileşmeler sağlamıştır. Arcuri ve Galeotti [161] test senaryosuna göre SQL verisini otomatik olarak oluşturan yeni bir yöntem önermiş ve kapsama miktarında önemli iyileşmeler elde ederek daha fazla sayıda hatayı tespit etmiştir. Arcuri [162] çalışmasında kara kutu testi için de bir model önermiş, 8 farklı RESTful web servisler üzerinde test etmiş ve kara kutu testi ile de pek çok hata bulmuştur. Viglianisi ve ark. [163] RESTful web servis testi için RESTTESTGEN isimli bir araç geliştirmiş ve yapmış oldukları testler sonucunda 87 farklı hatayı tespit ettiklerini belirtmiştir.

Yazılım testi üretiminde bu yöntemlerin önerilmesi büyük bir gelişme olmakla birlikte, arama algoritmalarının verimliliğinin artırılmasına yönelik daha etkin metotlara ihtiyaç duyulmaktadır. Nesne yönelimli programlama ve RESTful web servisler için test senaryosu üretiminde daha etkin performans gösteren algoritmaların geliştirilmesi

¹<https://github.com/EMResearch/EvoMaster>

oldukça önemlidir. Bu tez çalışmasının amacı bu gelişmeyi sağlamak için daha önce test verisi üretiminde etkinliği kanıtlanmış olan [103] ABC algoritmasının hem nesne yönelimli programlama hem de RESTful web servisler için test senaryosu üretimine uygun hale getirilmesidir.



3. BÖLÜM

NESNE YÖNELİMLİ PROGRAMLAMA İÇİN TEST SENARYOSU ÜRETİMİ

3.1. Giriş

Yazılımlar karmaşıktıkça ve boyutları arttıkça yapısal programlama yerine daha karmaşık eylemlerin ortaya daha rahat konabileceği nesne yönelimli programlama gibi paradigmlar oldukça popüler hale gelmiştir. Buna bağlı olarak, nesne yönelimli programlamanın; kalıtım, kapsülleme, çok-biçimlilik, soyutlama, bilgi gizleme gibi özellikleri sebebiyle test süreçleri de zorlaşmaktadır [8, 164]. Örneğin kalıtım, yeni bir alt sınıf test edildiğinde buna bağlı olarak diğer alt ve üst sınıfların tekrar tekrar test edilmesine sebep olmaktadır [8]. Farklı nesnelerin bütün durumlarının ve davranışlarının testi de bu sınıflara kısıtlı erişim sebebiyle zorlaşmaktadır. İlişkilendirme (association) bir başka sınıfın metoduna yapılan çağrılar test etmeyi zorlaştırmaktadır. Benzer şekilde çok-biçimlilik sebebiyle geçersiz kılınan (override) metotların testi de bir başka zorluk yaratan durumdur [164].

Tez çalışmasının bu bölümünde nesne yönelimli programlama için otomatik test senaryosu üretimine odaklanılmıştır. Nesne yönelimli programlama için test senaryoları çeşitli metot çağrılar ve bu çağrılara ait parametrelerden oluşmaktadır. Bunların doğru sıra ile oluşturulmasıyla test senaryoları meydana gelmektedir. Bu da kombinasyonel bir problem çeşidi olduğundan bu bölümde, iki farklı çok kriterli kombinasyonel ABC algoritması önerilmiştir. İlk olarak literatürde sıkça kullanılan çaprazlama tekniklerinin etkinliği incelenmiş ve uygun çaprazlama tekniği ABC algoritmasına entegre edilmiştir. Daha sonra ABC algoritmasını, yakınsama hızını iyileştirmek ve değerlendirme sayısını da etkin kullanmak adına kapsanan hedefleri tuttuğumuz arşiv mekanizması ile iyileştirilmiştir. Daha sonra ABC algoritmasındaki eksiklikler ele

alınarak, yeni bir ABC modeli önerilmiş ve Modifiye Edilmiş Arşiv Tabanlı Yapay Arı Koloni Algoritması (MABC) adı verilmiştir. Bu modelde, faydalanma ve keşif arasındaki dengeyi oluşturmak adına çeşitli modifikasyonlar gerçekleştirilmiştir. Görevli arı fazındaki yeni birey üretim mekanizmasında, seçilen çözümlerden biri tamamen rastgele olarak belirlenmiş ve çeşitlilik artırılmıştır. Gözcü arı fazında ise üretilen tüm çözümler bir havuza atılmaktadır ve daha sonra bu havuzdan iyi olan çözümler belirlenerek popülasyon tekrar oluşturulmaktadır. Bu modifikasyonların uygulandığı MABC algoritması ile literatürde sıkça kullanılan MGA, SGA ve RS algoritmaları kıyaslanmış, istatistiksel analizleri gerçekleştirilmiştir.

3.2. Nesne Yönelimli Test Senaryosu Üretimi

Bir test senaryosu, komutlardan oluşan metotların birleşiminden meydana gelen birim testlerin bütünüdür. Örnek bir sınıf ve bu sınıfa ait test senaryosu Şekil 3.1’de verilmiştir [165]. Fonksiyon çıktısının doğru ve geçerli olduğunu onaylamak için bir doğrulayıcı tarafından belirlenen girdiler fonksiyona verilmektedir [8]. Saydam kutu testinde ise bir doğrulayıcı olmaması durumunda en az test ile maksimum kapsamına erişme durumu optimizasyon problemi olarak ele alınabilmektedir [104].

```

public class Complex{
    public Complex log() {
        if(isNaN){
            return NaN;
        }
        return createComplex(
            FastMath.log(abs()),
            FastMath.atan2(imaginary, real)
        );
    }
    public Complex pow(double x){
        return this.log().multiply(x)
            .exp();
    }
    ...
}

@Test
public void test1() throws Throwable{
    Complex c0 = new Complex(Double.NaN);
    Complex c1 = c0.pow(Double.NaN);
    assertEquals(Double.NaN,
        c1.getArgument(), 0.01D);
}

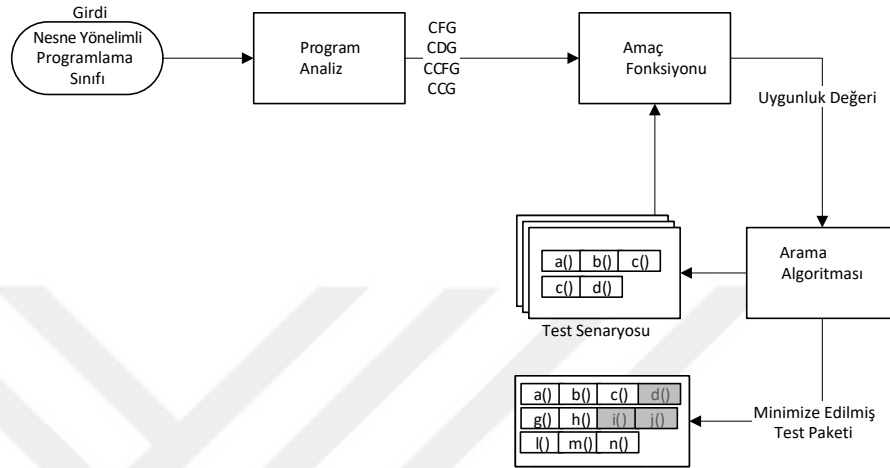
@Test
public void test2() throws Throwable{
    Complex c0 = Complex.ZERO;
    Complex c1 = c0.pow(c0);
    assertFalse(c1.isInfinite());
    assertTrue(c1.isNaN());
}

```

Şekil 3.1. Örnek bir sınıf ve bu sınıfa ait test senaryosu.

Nesne yönelimli programlama için arama tabanlı test senaryo üretim şeması Şekil 3.2’de verilmiştir. İlk olarak, sisteme test edilecek olan sınıf verilir. Program analiz birimi

ise bu sınıfı uygun gösterimlere çevirir. Daha sonra bu gösterimlere bağlı olarak arama algoritmalarının ürettiği test senaryolarının kalitesi belirlenir ve algoritmaya geri dönüş olarak verilir. Arama sonlandığında ise test senaryoları uygun hale getirilip son kullanıcıya sunulmaktadır.



Şekil 3.2. Genel arama tabanlı test senaryosu üretim şeması.

3.3. Program Analiz Birimi

Program analiz birimi almış olduğu kaynak kodu uygun gösterime çevirir. Bu gösterimler, üretilen testlerin kalitelerinin belirlenmesi için amaç fonksiyonu tarafından kullanılır. Kaynak kod bu birim tarafından, kontrol akış çizgesi (control flow graph, CFG), kontrol bağımlılık çizgesi (control dependence graph, CDG), sınıf çağrı çizgesi (class call graph, CCG) ve sınıf kontrol akış çizgesine (class control flow graph, CCFG) çevrilir. CFG program yapısının grafiksel bir gösterimidir ve yönlendirilmiş bir çizge ile tanımlanmaktadır. $G = (N, E, s, e)$ olmak üzere, N komutlardan sorumlu düğümü setini temsil ederken, E ise kenarları temsil etmektedir. s ve e ise tekil düğümlerdir ve s başlangıç düğümünü, e ise programın sonlandırıldığı bitiş düğümünü temsil etmektedir [57]. CDG bağımlılıkların gösterildiği bir çizgedir [166, 167]. Sınıfların birbirine bağımlılıklarını belirtmektedir. Kardeş düğümler bir ebeveyn düğüme aittir ve düğümlerden biri çalışırsa diğer tüm kardeş düğümler de çalışmaktadır. CCG her bir sınıftaki metod çağrı sıralarının tutulmuş olduğu çizgedir. CCFG bir çerçeve altına alınmış sınıfın veri akış analizini temsil etmektedir. Buradaki çerçeve erişilebilir (public) metotlara yapılan çağrı dizisini modellemektedir ve giriş, çıkış, döngü, çağrı ve geri dönüş

(return) çerçeveleri olmak üzere toplam beş düğümden oluşmaktadır. Bu düğümlerin bağlantıları da (giriş, döngü), (döngü, çağrı), (döngü, çıkış) ve (geri dönüş, döngü) olmak üzere dört farklı çizgeden oluşmaktadır [168, 169].

3.4. Temel Yapay Arı Koloni Algoritması

2005 yılında Karaboğa [170] tarafından literatüre sunulan Yapay Arı Koloni algoritması, arıların doğadaki yiyecek arama davranışını modellemektedir ve popülasyon tabanlı sezgisel algoritmalarından biridir. Bu algortmada arılar yiyecek aramadaki iş bölümüne göre, görevli, gözcü ve kâşif olmak üzere üç gruba ayrılmaktadır. Görevli arılar hafızalarında tuttıkları kaynaklar etrafında arama yapmaktadır. Gözcü arılar ise görevli arılardan aldıkları bilgilere göre daha iyi yiyecek kaynağı olma potansiyeli olan bölgelerde aramalarını sürdürmektedir. Kaşif arılar keşfedilmemiş bölgelerde rastgele arama yaparak global arama gerçekleştirmektedir. Yapay arı koloni algoritmasının temel adımları Algoritma 3.1’de verilmiştir.

- 1: Başlangıç popülasyonunun oluşturulması
- 2: Uygunluk değerinin hesaplanması
- 3: **repeat**
- 4: Görevli arı aşaması
- 5: Gözcü arı aşaması
- 6: Kâşif arı aşaması
- 7: En iyi değerin seçilmesi
- 8: **until** Durdurma kriteri sağlanana kadar

Algoritma 3.1: Yapay arı koloni algoritmasının temel adımları.

Yapay arı koloni algoritmasında ilk olarak kontrol parametreleri belirlenir ve ardından yiyecek kaynaklarının pozisyonlarına karşılık gelen başlangıç popülasyonu alt ve üst sınır aralığında rastgele olarak oluşturulur. Ardından popülasyon bireylerinin uygunluk değerleri hesaplanır ve görevli, gözcü ve kâşif arı fazları çalışmaya başlar. Görevli arı fazı, Eşitlik 3.1 kullanılarak, çözümlerin komşuluklarının araştırılmasını sağlamaktadır. Eşitlik 3.1’de verilen x' yeni aday çözüm vektörüne, ϕ , $[-1, 1]$ aralığında rastgele bir değere, x_k ise rastgele seçilmiş komşu vektörüne karşılık gelmektedir. Yeni çözüm üretildikten sonra aç gözlü seçim mekanizması kullanılarak yeni ve mevcut çözümler arasından daha iyi olan seçilir. Yeni çözüm, mevcut çözümden daha iyi uygunluk değerine sahip ise yeni çözüm seçilirken, mevcut çözümün daha iyi olması durumunda ise mevcut çözüm

popülasyonda korunur ve ilgili sayaç (*trial*) bir artırılır. Bu sayaç, yiyecek kaynağının tükenip tükenmediğine karar vermekte kullanılmaktadır.

$$x'_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) \quad (3.1)$$

Gözcü arı aşamasında da Eşitlik 3.1 kullanılarak yeni çözüm üretilmektedir. Hangi kaynağın seçileceğine karar vermek için ise iyi çözümlerin daha yüksek olasılıkla seçilmesini sağlayan olasılıksal seçim metodu kullanılmaktadır. Bu metot, iyi bireylerin seçilme olasılığını artırarak iyi çözümlerin etrafında yerel arama yapılmasına olanak tanırken, kalite değeri düşük çözümlerin etrafında araştırma yapılma şansını da tanımaktadır. Bireylerin olasılık hesaplamaları Eşitlik 3.12 ile yapılmaktadır.

$$p_i = \frac{uygunluk_i}{\sum_{i=1}^{NP} uygunluk_i} \quad (3.2)$$

Burada p_i , i . çözümün seçilme olasılığına ve $uygunluk_i$ çözümün kalitesine karşılık gelmektedir. Görevli arı fazında olduğu gibi burada da aç gözlü seçim mekanizması kullanılmaktadır ve iyileştirme gerçekleşmeyen çözümlerin ilgili sayaç (*trial*) değeri bir artırılmaktadır.

Arıların nektar toplaması sonucunda zamanla yiyecek kaynakları tükenmektedir. Bu durum da çözümün artık iyileştirilemiyor olmasına karşılık gelmektedir. Kaynağın tükenip tükenmediğinin belirlenmesi "*limit*" adı verilen kontrol parametresi ile belirlenmektedir. x_i konumundaki bir çözümün deneme sayacı *limit* geliştirileme sayısını aştığı zaman ilgili kaynak terk edilmekte ve o kaynağın arısı kâşif arı olarak rastgele araştırma gerçekleştirmektedir.

3.5. Arşiv Tabanlı Yapay Arı Koloni Algoritması (ABC)

Test senaryosu üretiminin kombinasyonel bir problem olması ve temel ABC algoritmasının sürekli uzayda çalışmasından dolayı ABC algoritmasının ayrık uzayda çalışabilecek şekilde düzenlenmesi gerekmektedir. Bu nedenle ABC algoritmasının yeni çözüm mekanizması ayrık uzaya uygun olarak güncellenmiştir. Ayrıca yakınsama

hızını artırmak için ABC algoritmasına arşiv mekanizması entegre edilmiştir. ABC algoritmasının nesne yönelimli programlama için uygun hale getirilmiş versiyonu Algoritma 3.2’de verilmiştir ve ilerleyen alt bölümlerde bu algoritmanın ayrıntıları anlatılmıştır.

3.5.1. Temsil Biçimi

Test üretiminin kombinasyonel bir problem olması nedeniyle sayısal optimizasyon için önerilen yapay zeka algoritmalarının gösterim ve yeni çözüm üretim mekanizmaları, problemin doğasına uygun olarak yeniden tanımlanmalıdır. Her bir çözüm ($T = \{t_1, t_2, \dots, t_n\}$) olarak temsil edilir. Burada (t_i) çağrılan metot ile giriş parametrelerini temsil eden test senaryolarıdır ve $n \in [1, N]$ komutundaki N tüm olası test senaryo setlerini temsil eder.

Bir test senaryosu (t), komutların (s) sıra ile bir araya gelmesinden oluşmaktadır ($l_j \in [1, L]$, $t_j = \langle s_1, s_2, \dots, s_{l_j} \rangle$). Test senaryosunun uzunluğu ilgili senaryoda bulunan komutların toplam uzunluğudur ($|T| = \sum_{t_j \in T} l_j$). Hem n hem de l değişken boyutlu olduğundan gösterimin boyutu çevrimler boyunca değişken uzunluktadır.

Her bir komutun ($v(s_i)$) bir türü $\tau(v(s_i))$ bulunmaktadır. Bu türler; temel komutlar, yapıcı komutlar, alan komutları, metot komutları ve atama komutları olarak ayrılmaktadır [171, 172] ve aşağıda ayrıntıları verilmiştir.

- **Temel Komutlar (primitive statements):** *double deger = 1.0* gibi integer, boolean, string, array değişkenlerini ifade etmektedir. Herhangi bir tipte dizi tanımlamaları da temel komutlar kategorisine girmektedir (*Sinif[] n1 = new Sinif[3]* gibi).
- **Yapıcı Komutlar (constructor statements):** Verilen sınıfın yeni bir örneğini oluşturan komutlardır. Örneğin *Sinif n1 = new Sinif()* gibi bir tanım yapıcı komuttur ve test aracı, yapıcı komutların girdi parametrelerini ve türünü tanımlayıp saklamalıdır.
- **Alan Komutları (field statements):** Nesnelerin public değişkenlerine erişim sağlamaktadır. Örneğin, *int boyut = n1.boyut* komutu bir alan komutudur. Burada komutun tipi ve değeri kontrol edilmelidir.

Giriş: MCN : Maksimum Döngü sayısı, δ : Amaç fonksiyonu, CS : Popülasyon büyüklüğü, c_f : Çaprazlama fonksiyonu, c_p : Çaprazlama oranı, m_f : Mutasyon fonksiyonu, m_p : Mutasyon oranı, $Limit$: Tükenmiş kaynakları belirlemek için maksimum deneme sayısı

```

1:  $trial_i \leftarrow 0, i = 1 \dots p_s$ 
2:  $P \leftarrow RastgeleYiyecekKaynagiUret(CS)$ 
3:  $AmacFonksiyonu(\delta, P)$ 
4:  $dongu \leftarrow 0$ 
5: while  $dongu < MCN$  do
6:   //Görevli Arı Fazı
7:    $i \leftarrow 0$ 
8:   while  $i < CS$  do
9:      $p_1 \leftarrow P_i$ 
10:     $p_2 \leftarrow P_{rand}$ 
11:     $o_1, o_2 \leftarrow Caprazlama(c_f, c_p, p_1, p_2)$ 
12:     $o_1 \leftarrow Mutasyon(m_f, m_p, o_1)$ 
13:     $o_2 \leftarrow Mutasyon(m_f, m_p, o_2)$ 
14:     $AmacFonksiyonu(\delta, o_1)$ 
15:     $AmacFonksiyonu(\delta, o_2)$ 
16:    if  $EnIyi(o_1, o_2), p_1$ 'den daha iyi ise then
17:       $p_1 \leftarrow EnIyi(o_1, o_2)$ 
18:    else
19:       $DenemeSayaciniArtir(p_1)$ 
20:    end if
21:     $i++$ 
22:  end while
23:  //Gözcü Arı Fazı
24:   $prob \leftarrow OlasilikHesapla(P)$ 
25:   $i \leftarrow 0$ 
26:  while  $t < CS$  do
27:    if  $rand < prob_i$  then
28:       $t++$ 
29:       $p_1 \leftarrow P_i$ 
30:       $p_2 \leftarrow P_{rand}$ 
31:       $o_1, o_2 \leftarrow Caprazlama(c_f, c_p, p_1, p_2)$ 
32:       $o_1 \leftarrow Mutasyon(m_f, m_p, o_1)$ 
33:       $o_2 \leftarrow Mutasyon(m_f, m_p, o_2)$ 
34:       $AmacFonksiyonu(\delta, o_1)$ 
35:       $AmacFonksiyonu(\delta, o_2)$ 
36:      if  $EnIyi(o_1, o_2), p_1$ 'den daha iyi ise then
37:         $p_1 \leftarrow EnIyi(o_1, o_2)$ 
38:      else
39:         $DenemeSayaciniArtir(p_1)$ 
40:      end if
41:    end if
42:     $i \leftarrow ++i \bmod CS$ 
43:  end while
44:  //Kaşif Arı Fazı
45:   $i \leftarrow 0$ 
46:  while  $i < CS$  do
47:    if  $limit_{P_i} > Limit$  then
48:       $p_i \leftarrow YeniBirey()$ 
49:    end if
50:  end while
51:   $ArsivVeAmacFonksiyonDegerleriniGuncelle()$ 
52:   $dongu \leftarrow dongu + 1$ 
53: end while

```

Algoritma 3.2: ABC algoritmasının sözde kodları

- **Metot Komutları (method statements):** Nesnelerin non-statik veya sınıfların statik metotlarının çağrılmasını sağlayan komut tipidir. Örneğin, *int m = nl.metot()* metot komutudur. Tür ve döndürülen değerin kontrol edilmesi gerekmektedir.
- **Atama Komutları (assignment statements):** Atama komutları nesnelerin public alanlarının veya dizilerin değer atamasında kullanılan komut türüdür. *nl[0]=new Sinif()* veya *nl.deger = 3* gibi bir atama, atama komutu olarak adlandırılmaktadır.

3.5.2. Amaç Fonksiyonları

Amaç fonksiyonu, algoritmaları arama uzayında daha iyi çözümlerin bulunduğu bölgelere yönlendirmeyi hedeflemektedir. Yeni çözümler amaç fonksiyonundan elde edilen değere göre oluşmaktadır. Test üretiminde amaç fonksiyonu tek bir kriter olarak tanımlanabildiği gibi çoklu kriter olarak da tanımlanabilmektedir. Bu tez çalışmasında aşağıda tanımlanan amaç fonksiyonları kullanılmıştır [165].

3.5.2.1. Metot Kapsamı (Method Coverage)

Metot Kapsamı mevcut metotlardan kaçının kapsanabildiğini belirten bir metriktir. Eşitlik 3.3 ile hesaplanmaktadır. $|\cdot|$ operatörü toplam sayıyı ifade etmektedir.

$$f_{MC}(TestPaketi) = |ToplamMetot| - |KapsananMetot| \quad (3.3)$$

3.5.2.2. Satır Kapsamı (Line Coverage)

Eşitlik 3.4 ile hesaplanabilen satır kapsamı yorum satırı olmayan satırlardan ne kadarının kapsanabildiğini gösteren metriktir.

$$f_{LC}(TestPaketi) = v(|NCLs| - |KapsananSatirlar|) + \sum_{b \in BCD} v(d_{\min}(b, Suite)) \quad (3.4)$$

$|NCL|$ yorum satırı olmayan toplam satır sayısını, $v(x)$ ise dönen değeri $[0, 1]$ aralığına normalize edecek normalizasyon fonksiyonunu belirtmektedir.

$d_{\min}(b, Suite)$, dallar kümesindeki $((B_{CD}))$ her dal b için gerçekleştirilen tüm yürütmeler içindeki minimum dal mesafesini vermektedir.

3.5.2.3. Dal Kapsamı (Branch Coverage)

Dal kapsamı, test senaryosu olarak temsil edilen hedef t ve kontrol akış (x) şemalarındaki bağımsız kenarların minimum sayısı olan dal uzaklığı ile $(d(t, x))$ hesaplanmaktadır ve Eşitlik 3.5'te verilmiştir.

$$f_{BC}(Suite) = \sum_{b \in B} v(d(b, Suite)) \quad (3.5)$$

SUT içerisindeki bütün dallar B ile temsil edilirken, $d(b, Suite)$ ise Eşitlik 3.6 ile tanımlanmıştır.

$$d(b, Suite) = \begin{cases} 0 & \text{Dal kapsanırsa,} \\ v(d_{\min}(b, Suite)) & \text{Karşılaştırma belirtimi en az} \\ 1 & \text{iki sefer çalışırsa,} \\ & \text{aksi takdirde.} \end{cases} \quad (3.6)$$

3.5.2.4. Doğrudan Dal Kapsamı (Direct Branch Coverage)

Yalnızca doğrudan metot çağrılarının dikkate alındığı dal kapsamı metriğidir. Dolaylı çağrılar dikkate alınması durumunda test aktiviteleri zorlaşmaktadır.

3.5.2.5. Çıkış Kapsamı (Output Coverage)

Çıkış kapsamı girişe bağlı olarak elde edilebilecek çıkış değerleri ile hesaplanmaktadır ve Eşitlik 3.7'de verilmiştir.

$$f_{OC}(Suite) = \sum_{g \in G} v(d_o(g, Suite)) \quad (3.7)$$

G çıkış hedef setini temsil ederken, $d_o(g, Suite)$ çıktı mesafe fonksiyonudur.

3.5.2.6. Zayıf Mutasyon (Weak Mutation)

Mutasyon testinde kapsama, kodda küçük değişiklikler ve yapay hatalarla oluşturulan mutant ile orijinal kod arasındaki farklılığa göre Eşitlik 3.8 ile hesaplanmaktadır.

$$f_{WM}(Suite) = \sum_{\mu \in M_C} d_w(\mu, Suite) \quad (3.8)$$

$d_w(\mu, Suite)$ mutant ve orijinal kod arasındaki mesafeyi ölçen ve Eşitlik 3.9 ile tanımlanan fonksiyondur ve M_C mutant setini belirtmektedir.

$$d_w(\mu, Suite) = \begin{cases} 1 & \text{mutant } \mu \text{ erişilemediyse,} \\ v(d_{min}(\mu, Suite)) & \text{mutant } \mu \text{ erişildiyse.} \end{cases} \quad (3.9)$$

3.5.2.7. İstisna Kapsamı (Exception Coverage)

İstisna kapsamı her bir sınıftaki tanımlı tekil istisnaların gerçekleşmesi ile Eşitlik 3.10 kullanılarak hesaplanan bir metriktir.

$$f_{EC}(Suite) = \frac{1}{1 + N_E} \quad (3.10)$$

N_E istisna sayısını vermektedir.

3.5.2.8. Çok Kriterli Amaç Fonksiyonu

Amaç fonksiyonu, aramayı daha iyi bölgelere doğru yönlendirerek algoritmaya kılavuzluk etmektedir. Test senaryosu üretiminde amaç fonksiyonu sadece bir kapsama metriği ile tanımlanabildiği gibi birden fazla kapsama metriği de dikkate alınabilmektedir. Önerilen ABC algoritmasında, bütün kapsama metriklerinin aynı anda dikkate alındığı çok kriterli yaklaşım kullanılmıştır. Bölüm 3.5.2’de tanımlanmış olan amaç fonksiyonları w_i cezalandırma faktörü kullanılarak Eşitlik 3.11 ile hesaplanmaktadır.

$$f_{combined} = \sum_{i=1}^n w_i f_i \quad (3.11)$$

3.5.3. Başlangıç

ABC algoritmasının başlangıç aşamasında yiyecek kaynağı sayısı (CS), maksimum döngü sayısı (MCN) ve *limit* değerlerini içeren kontrol parametreleri atanmaktadır. CS boyutlu popülasyon rastgele çözümlerden ($X = T_1, T_2, \dots, T_{CS}$) oluşturulmaktadır. Her bir çözümdeki test senaryo sayısı (n) bu çözüm oluşturulurken $[1, N]$ aralığında rastgele olarak belirlenmektedir. Her bir test senaryosu ise $[1, L]$ aralığında rastgele olarak belirlenen komut sayısına (l_j) sahiptir. N ve L ile bir çözümün sınırlandırılması sayesinde çözümlerin aramayı imkansız hale getirecek şekilde büyümesi problemi olan bloat problemi çözülmektedir. Bu kısıtları sağlayan test senaryoları oluşturulurken başta boş bir test senaryosu oluşturulmakta ve uzunluğu l_j olana kadar komut ekleme operatörü çalıştırılmaktadır.

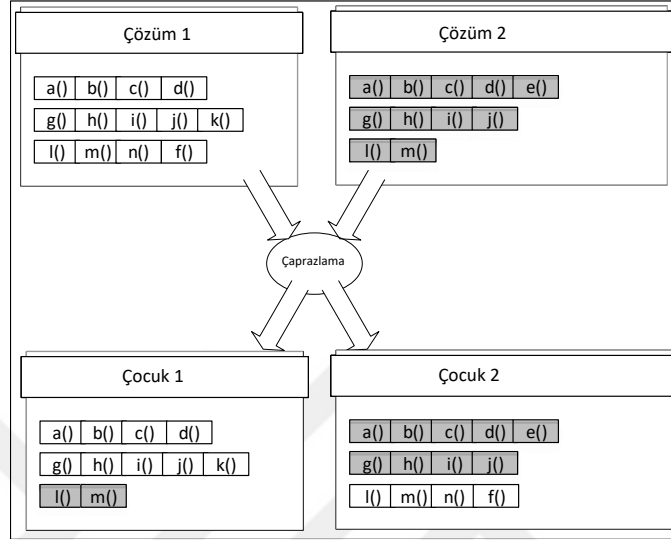
3.5.4. Çaprazlama ve Mutasyon

Test senaryosu üretiminde kullanılan çaprazlama işlemi için literatürde genellikle üç farklı çaprazlama metodu kullanılmaktadır. Kullanılan çaprazlama teknikleri;

- Tek noktalı çaprazlama (Single Point, SP)
- Sabit tek noktalı çaprazlama (Single Point Fixed, SPF)
- Bağıl tek noktalı çaprazlama (Single Point Relative, SPR)

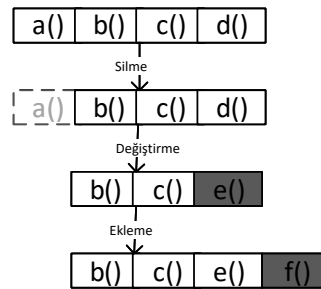
olarak sıralanmaktadır. SP her birey için birer çaprazlama noktası rastgele olarak belirlenir. Belirlenen noktalar kullanılarak çaprazlama işlemi gerçekleştirilir. SPF için ortak bir çaprazlama noktası belirlenir ve bu ortak noktadan bireyler çaprazlanır. SPR uygulanırken $(0, 1)$ aralığında ayırım oranı belirlenir. Daha sonra; $AyrimNoktasi1 = \lfloor ParentSize1 * AyrimOrani \rfloor + 1$ ve $AyrimNoktasi2 = \lfloor ParentSize2 * AyrimOrani \rfloor + 1$ olmak üzere 2 farklı ayırım noktası hesaplanır. Oluşturulacak olan yeni bireyin ilk kısmı seçilen birinci bireyin 0. elemanından $AyrimNoktasi1$ 'e kadar olan kısımdan ve ikinci bireyin $AyrimNoktasi2$ 'den son elemanına kadar olan kısımdan oluşmaktadır. İkinci birey ise 2. bireyin 0. elemanından $AyrimNoktasi2$ 'ye kadar olan

kısmı ile AyrimNoktasi1'den son elemana kadar olan test senaryoları alınmaktadır. Daha sonra bu iki birey karşılaştırılarak iyi olan birey popülasyona katılmaktadır. Çaprazlama operatörüne ait örnek Şekil 3.3'te verilmiştir.



Şekil 3.3. Çaprazlama operatörünün çalışmasına örnek.

Mutasyon işlemi, test senaryolarının silinmesi, değiştirilmesi veya eklenmesi olarak gerçekleştirilmektedir. Silme, değiştirme ve ekleme işlemlerinde çözümdeki herhangi bir komut $1/|T|$ olasılığına bağlı olarak sırasıyla silinir, değiştirilir veya yeni komut eklenir. Buradaki $|T|$, ilgili senaryoda bulunan komutların toplam uzunluğudur. Bu eylemlere ait örnek Şekil 3.4'te verilmiştir.



Şekil 3.4. Mutasyon operatörünün çalışmasına örnek.

3.5.5. Görevli Arı Fazı

Yeni çözüm üretilirken, ayrık gösterime uygun olarak aritmetik komşu seçim mekanizması yerine çaprazlama ve mutasyon operatörlerinin kullanıldığı bir mekanizma ile yeni komşu çözümler üretilmektedir. Literatürde pek çok çaprazlama ve mutasyon

operatörü bulunmaktadır [173]. Bu çalışmada Bölüm 3.5.4'te belirtilen operatörler kullanılmıştır.

Görevli arı fazında her bir çözüm için bir komşu çözüm seçilmektedir. Daha sonra bu çözüme, komşu çözüm ile çaprazlama ve ardından oluşan iki çocuk çözüme mutasyon operatörleri uygulanmaktadır. Ardından aç gözlü seçim ile, yeni oluşturulan bu iki çözümün uygunluk değeri mevcut çözümden daha iyi ise popülasyona katılmakta, değilse popülasyonda kalmaya devam etmekte ve ilgili sayaç (*trial*) değeri bir artırılmaktadır.

3.5.6. Gözcü Arı Fazı

Gözcü arı fazında arıların daha iyi nektar miktarına sahip çözümlere yönelmesi sağlanmaktadır. Bu sebeple yüksek nektar miktarına sahip yiyecek kaynaklarının seçilme ihtimali daha yüksek olmaktadır. Bunu sağlamak için her bir x_i çözümünün uygunluk değerlerine bağlı olarak seçilme olasılığı, p_i , değerleri Eşitlik 3.12 kullanılarak hesaplanmaktadır.

$$p_i = \frac{fitness_i}{CS \sum_{j=1} fitness_j} \quad (3.12)$$

Bir çözüm $[0, 1]$ aralığında rastgele belirlenen değer ile, stokastik olarak olasılık değerine bağlı seçilmektedir. Eğer rastgele seçilen sayı ilgili çözümün olasılık değerinden küçük ise o çözüm komşu olarak seçilmektedir. Ardından, görevli arı fazında olduğu gibi rastgele bir komşu belirlenir ve bu çözümler kullanılarak çaprazlama operatörü uygulanır. Daha sonra oluşan iki çözüme de mutasyon operatörleri uygulanır ve bu çözümler arasında aç gözlü seleksiyon uygulanarak en iyi çözüm belirlenir. Yeni üretilen çözüm daha iyi değilse ilgili sayaç (*trial*) değeri bir artırılır.

3.5.7. Kâşif Arı Fazı

Kâşif arı fazı temel ABC'de olduğu gibi işlemektedir. Sayaç (*trial*) değeri kontrol parametresi olan *limit* değerini aştığında o çözüm bırakılmakta ve yerini yeni bir çözüm almaktadır.

3.5.8. Seçim Stratejisi

Test senaryosu üretiminde üretilen testlerin boyutlarının çok fazla artması sebebiyle testin çalıştırılmayacak hale gelmesi bloat problemi olarak adlandırılmaktadır. Bu problemi engellemek adına amaç fonksiyon değerinin yanında taşma (bloat) kısıtlamalarını da sağlayıp sağlamadığı kontrol edilmektedir. Eğer üretilen çözüm bloat kısıtını sağlamıyor ise bu çözüm her durumda daha kötü kabul edilmektedir. Bu kısıt ise maksimum test senaryo sayısı (N) ve maksimum komut sayısının (L) limitlerinin belirlenmesi ile sağlanmaktadır.

3.5.9. Arşiv

Test senaryosu üretiminde bir hedefi kapsayan test senaryosu üretilse bile, diğer kapsama hedeflerine erişebilmek için arama devam etmektedir. Bu da sistem kaynaklarının hali hazırda kapsanmış hedefler için harcanmasına sebep olabilmektedir. Kaynakların kapsanmamış hedeflere odaklanmasını sağlamak için bir hedefi kapsayan test senaryosunun arşive atılıp o hedefin de amaç fonksiyonu hesaplamasından çıkarılması gerekmektedir [151]. Önerilen yaklaşım da kapsanmış olan hedeflere ait test senaryolarını arşive atarak kapsanmamış hedeflerin keşfine odaklanmayı hedeflemektedir. Her bir iterasyonun sonunda amaç fonksiyonu (Eşitlik 3.11) kapsanmış hedefleri yok sayarak güncellenir.

3.5.10. Budama

Test senaryosu üretildikten sonra, geri kalan tüm komutlar kapsama miktarına katkıda bulunana kadar her bir gereksiz ifade teker teker silinir [174]. Bu adım, hem test senaryolarının sayısının hem de uzunluklarının [171] azalmasını sağlar.

3.6. Modifiye Edilmiş Arşiv Tabanlı Yapay Arı Koloni Algoritması (MABC)

ABC algoritmasının keşif ve faydalanma becerilerini iyileştirmek için Modifiye Edilmiş Arşiv Tabanlı Yapay Arı Koloni Algoritması (MABC) önerilmiştir. MABC algoritmasının adımları Algoritma 3.3'te verilmiştir. Görevli ve gözcu arı fazlarında

çeşitli modifikasyonlar gerçekleştirilmiştir. Görevli arı fazındaki yeni birey üretim mekanizması çeşitliliği artırmak için değiştirilmiştir. Yeni birey üretilirken seçilen çözümlerden biri i . birey olurken, diğer birey tamamen rastgele seçilmektedir. Bu değişiklik sayesinde popülasyondaki çeşitlilik artmaktadır. Gözcü arı fazında ise çaprazlama ve mutasyon operatörleri ile üretilen bütün bireyler bir havuza atılmaktadır. Ardından en iyi CS çözüm havuzdan seçilmektedir. Bu sayede yeni üretilen çözüm ebeveyn çözümden daha kötü fakat popülasyondakilerden daha iyiye popülasyona dahil edilmektedir. Bu da algoritmanın faydalanma becerisini geliştirmektedir. Bu değişikliğin sebep olduğu çeşitliliğin azalması görevli arı fazındaki değişiklik ile dengelenmektedir.

3.7. Literatürde Bulunan Algoritmalar

Nesne yönelimli programlama için test senaryosu üretiminde yaygın olarak kullanılan algoritmalar olan RS, SGA ve MGA algoritmalarının ayrıntıları bu bölümde anlatılmıştır.

3.7.1. Rastgele Arama

RS basitliğinin yanında sezgisel olarak da ilgi çekici bir teknik olarak kabul edilmektedir [37, 175]. RS’de test senaryoları düzgün dağılıma göre rastgele oluşturulur. Eğer bir test senaryosu test paketinin kapsamını yükseltirse, bu test paketine dahil edilmektedir, aksi takdirde test paketinden çıkarılmaktadır. Bu süreçte herhangi bir hedefi kapsayan test senaryosu, test paketine eklendiği için arşiv mekanizması kullanılmamaktadır.

RS ile test senaryosu üretimi oldukça basittir ve birçok problemde uygun maliyetle yeterli kapsamı sağlayabildiği gösterilmiştir [103, 176]. Karmaşık yazılımlar veya karmaşık kriterlerde RS başarısız olsa da uygulama kolaylığı sebebiyle popüler bir tekniktir.

3.7.2. Standart Genetik Algoritma

Genetik algoritma pek çok alanda oldukça yoğun bir şekilde kullanılmaktadır. Uygulanması kolaydır ve genelde iyi sonuç elde etmektedir. Standart Genetik Algoritma’nın (SGA) temel adımları Algoritma 3.4’te verilmiştir [151]. Algoritma ilk olarak rastgele popülasyon üreterek başlamaktadır (Satır 1). Ardından seçim stratejisi

Giriş: *MCN*: Maksimum Döngü sayısı, δ : Amaç fonksiyonu, *CS*: Popülasyon büyüklüğü, c_f : Çaprazlama fonksiyonu, c_p : Çaprazlama oranı, m_f : Mutasyon fonksiyonu, m_p : Mutasyon oranı, *Limit*: Tükenmiş kaynakları belirlemek için maksimum deneme sayısı

```

1:  $trial_i \leftarrow 0, i = 1 \dots p_s$ 
2:  $P \leftarrow RastgeleYiyecekKaynagiUret(CS)$ 
3:  $AmacFonksiyonu(\delta, P)$ 
4:  $dongu \leftarrow 0$ 
5: while  $dongu < MCN$  do
6:   //Görevli Arı Fazı
7:    $i \leftarrow 0$ 
8:   while  $i < CS$  do
9:      $p_1 \leftarrow P_i$ 
10:     $p_2 \leftarrow YeniBirey()$  //MODİFİKASYON
11:     $o_1, o_2 \leftarrow Caprazlama(c_f, c_p, p_1, p_2)$ 
12:     $o_1 \leftarrow Mutasyon(m_f, m_p, o_1)$ 
13:     $o_2 \leftarrow Mutasyon(m_f, m_p, o_2)$ 
14:     $AmacFonksiyonu(\delta, o_1)$ 
15:     $AmacFonksiyonu(\delta, o_2)$ 
16:    if  $EnIyi(o_1, o_2)$ ,  $p_1$ 'den daha iyi ise then
17:       $p_1 \leftarrow BEST(o_1, o_2)$ 
18:    else
19:       $DenemeSayaciniArtir(p_1)$ 
20:    end if
21:     $i++$ 
22:  end while
23:  //Gözcü Arı Fazı
24:   $prob \leftarrow OlasilikHesapla(P)$ 
25:   $i \leftarrow 0$ 
26:  while  $t < CS$  do
27:    if  $rand < prob_i$  then
28:       $t++$ 
29:       $p_1 \leftarrow P_i$ 
30:       $p_2 \leftarrow P_{rand}$ 
31:       $o_1, o_2 \leftarrow Caprazlama(c_f, c_p, p_1, p_2)$ 
32:       $o_1 \leftarrow Mutasyon(m_f, m_p, o_1)$ 
33:       $o_2 \leftarrow Mutasyon(m_f, m_p, o_2)$ 
34:       $AmacFonksiyonu(\delta, o_1)$ 
35:       $AmacFonksiyonu(\delta, o_2)$ 
36:       $DenemeSayaciniArtir(p_1)$ 
37:       $DenemeSayaciniArtir(p_2)$ 
38:       $DenemeSayaciniSifirla(o_1)$ 
39:       $DenemeSayaciniSifirla(o_2)$ 
40:       $P \leftarrow \{P\} \cup (o_1, o_2)$  //MODİFİKASYON
41:    end if
42:     $i \leftarrow ++i \bmod CS$ 
43:  end while
44:  //Yeni popülasyonun havuzdan seçim ile oluşturulması.
45:   $P \leftarrow AyniBireyleriTemizle(P, CS)$  //MODİFİKASYON
46:   $P \leftarrow EnIyiNBirey(P, CS)$  //MODİFİKASYON
47:  //Kâşif Arı Fazı
48:   $i \leftarrow 0$ 
49:  while  $i < CS$  do
50:    if  $limit_{P_i} > Limit$  then
51:       $p_i \leftarrow YeniBirey()$ 
52:    end if
53:  end while
54:   $ArsivVeAmacFonksiyonDegerleriniGuncelle()$ 
55:   $dongu \leftarrow dongu + 1$ 
56: end while

```

Algoritma 3.3: MABC algoritmasının sözde kodları

s_f ile iki birey seçilmektedir. Ardından bu iki birey kullanılarak çaprazlama operatörü c_f belli bir c_p çaprazlama oranı ile uygulanır ve o_1 ve o_2 olmak üzere iki yeni birey elde edilir (Satır 7). Ardından her iki bireye de mutasyon operatörü m_f , m_p olasılığı ile uygulanır. Bu oran m_p genellikle $1/n$ 'e eşitti ve n de kromozomdaki gen sayısıdır. Ardından bu mutasyona uğramış iki birey popülasyona dahil edilir (Satır 10). Her bir iterasyon sonunda ise bütün bireylerin amaç fonksiyon değerleri hesaplanır (Satır 13).

SGA bütün test kriterlerinin aynı anda dikkate alındığı yaklaşım olan tam test senaryosu üretimi için kullanılabilmektedir [165]. Ayrıca arşiv mekanizması da kullanılarak performansında iyileştirmeler sağlanmıştır [152].

Giriş: C : Durdurma kriteri, δ : Amaç fonksiyonu, PS : Popülasyon büyüklüğü, s_f : Seçim fonksiyonu, c_f : Çaprazlama fonksiyonu, c_p : Çaprazlama oranı, m_f : Mutasyon fonksiyonu, m_p : Mutasyon oranı

Çıktı: Optimize edilmiş P popülasyonu

```

1:  $P \leftarrow \text{RastgelePopulasyonUret}(PS)$ 
2:  $\text{AmaçFonksiyonu}(\delta, P)$ 
3: while  $\neg C$  do
4:    $N_p \leftarrow \{\} \cup \text{ELIT}(P)$ 
5:   while  $|N_p| < PS$  do
6:      $p_1, p_2 \leftarrow \text{SECIM}(s_f, P)$ 
7:      $o_1, o_2 \leftarrow \text{CAPRAZLAMA}(c_f, c_p, p_1, p_2)$ 
8:      $o_1 \leftarrow \text{MUTASYON}(m_f, m_p, o_1)$ 
9:      $o_2 \leftarrow \text{MUTASYON}(m_f, m_p, o_2)$ 
10:     $N_p \leftarrow N_p \cup \{o_1, o_2\}$ 
11:   end while
12:    $P \leftarrow N_p$ 
13:    $\text{AmaçFonksiyonu}(\delta, P)$ 
14: end while
15: return  $P$ 

```

Algoritma 3.4: SGA algoritmasının sözde kodları

3.7.3. Monotonik Genetik Algoritma

Monotonik Genetik Algoritma (Monotonic Genetic Algorithm, MGA) SGA algoritmasının varyantıdır. Algoritmanın temel adımları Algoritma 3.5'te verilmiştir [151]. İlk olarak rastgele popülasyon üretilmektedir (Satır 1). Ardından p_1 ve p_2 olmak üzere iki ebeveyn çözüm seçilmektedir (Satır 6). Ardından bu iki çözümden çaprazlama operatörü kullanılarak iki yeni çocuk çözüm üretilir. SGA her iki çocuk çözümü popülasyona katarken, MGA her bir çocuğu mutasyona uğrattıktan ve değerlendirdikten sonra, bir sonraki popülasyona en iyi çözümü içeren çocuk çiftini veya ebeveyn çiftini dahil etmektedir (Satır 12-16).

MGA bütün test kriterlerinin aynı anda dikkate alındığı yaklaşım olan tam test senaryosu üretimi için kullanılabilmektedir [165]. Ayrıca arşiv mekanizması da kullanılarak performansında iyileştirmeler sağlanmıştır [152].

Giriş: C : Durdurma kriteri, δ : Amaç fonksiyonu, PS : Popülasyon büyüklüğü, s_f : Seçim fonksiyonu, c_f : Çaprazlama fonksiyonu, c_p : Çaprazlama oranı, m_f : Mutasyon fonksiyonu, m_p : Mutasyon oranı

Çıktı: Optimize edilmiş P popülasyonu

```

1:  $P \leftarrow RastgelePopulasyonUret(PS)$ 
2:  $AmacFonksiyonu(\delta, P)$ 
3: while  $\neg C$  do
4:    $N_p \leftarrow \{\} \cup ELIT(P)$ 
5:   while  $|N_p| < PS$  do
6:      $p_1, p_2 \leftarrow SECIM(s_f, P)$ 
7:      $o_1, o_2 \leftarrow CAPRAZLAMA(c_f, c_p, p_1, p_2)$ 
8:      $o_1 \leftarrow MUTASYON(m_f, m_p, o_1)$ 
9:      $o_2 \leftarrow MUTASYON(m_f, m_p, o_2)$ 
10:     $AmacFonksiyonu(\delta, o_1)$ 
11:     $AmacFonksiyonu(\delta, o_2)$ 
12:    if  $EnIyi(o_1, o_2), EnIyi(p_1, p_2)$ 'den daha iyiye then
13:       $N_p \leftarrow N_p \cup \{o_1, o_2\}$ 
14:    else
15:       $N_p \leftarrow N_p \cup \{p_1, p_2\}$ 
16:    end if
17:  end while
18:   $P \leftarrow N_p$ 
19: end while
20: return  $P$ 

```

Algoritma 3.5: MGA algoritmasının sözde kodları

3.8. Deneysel Çalışmalar ve Bulgular

Deneylerin ilk kısmında literatürde test senaryosu üretimi için önerilmiş çaprazlama modellerinin etkinliği incelenmiştir. Deneylerin sonraki kısmında, arşivin etkisini incelemek için ABC algoritmasının arşivli ve arşivsiz versiyonları kıyaslanmıştır. Daha sonra etkin test senaryosu üretimi için önerilen MABC algoritmasının performansı çeşitli problem zorluk derecelerine göre incelenmiştir. Ardından literatürdeki popüler algoritmalar olan MGA, SGA ve RS algoritmaları ile MABC algoritması kıyaslanmış ve istatistiksel analizler gerçekleştirilmiştir. Son olarak algoritmaların yakınsama ve çeşitlilik analizleri verilmiştir.

3.8.1. Deney Konfigürasyonları

Deneylerde Java sınıflarından oluşan benchmark seti olan SF110 corpus problem seti

[177] kullanılmıştır. SF110 corpus problem seti 110 projede 20000'den fazla sınıfı barındırmaktadır.

Çaprazlama operatörünün etkisini irdelemek amacıyla literatürde en çok kullanılan çaprazlama teknikleri olan SPR, SP ve SPF tekniklerinin performansı incelenmiştir. Çaprazlama operatörlerinin incelenmesi, SF110 setinden rastgele seçilmiş dal sayısı $[2, 504]$ ve satır sayısı $[12, 661]$ aralığında değişen toplam 43 (Tablo 3.1) sınıf ile gerçekleştirilmiştir. ABC algoritması, her bir çaprazlama tekniği için her bir sınıf üzerinde ayrı ayrı 30'ar defa çalıştırılmıştır. Tüm çalıştırmalar 2 dakika boyunca sürmüştür. Popülasyon büyüklüğü 50 ve *limit* değeri 15 olarak tercih edilmiştir. Bütün çalıştırmaların hesaplama zamanı $43 * 30 * 2 = 2580$ dakikadır. Elde edilen sonuçlar ilk satırda ortalama ve standart sapma, ikinci satırda ortanca değer olarak verilmiştir.

Arşiv mekanizmasının etkinliği ve önerilen algoritmanın performansını incelemek için SF110 corpus problem setinden rastgele 6569 sınıf seçilmiştir. Seçilen bu sınıflar, ck tool¹ isimli araç kullanılarak analiz edilmiş, döngüsel karmaşayı belirten ve bir sınıf içindeki metotların McCabe döngüsel karmaşıklık değerlerini veren WMC metriğine göre çeşitli zorluk seviyelerine uygun olarak bölünmüştür. Bu zorluk seviyeleri Şekil 3.5'te gösterilen Histogram grafiğine göre gerçekçi ve dengeli dağılıma (realistic and balanced distributon) uygun, beş farklı seviyeye olarak belirlenmiştir. Tablo 3.2'de her bir seviyeye ait toplam sınıf sayısı ve bu sınıfların ortalama metrik değerleri verilmiştir. Tablo 3.2'deki metriklerin açıklamaları ise Tablo 3.3'te verilmiştir. Ortalama $\pm$ standart sapma değerleri ilk satırda, alt satırda parantez içinde ise ortanca değerleri verilmiştir. Algoritmalar %100 kapsama miktarına erişene veya 12500 değerlendirmeye sayısına ulaşana kadar çalıştırılmıştır. Bu iki bölümde maksimum değerlendirme sayısının dikkate alınma sebebi farklı özelliklere sahip farklı makinalarda çalışmaların gerçekleştirilmiş olmasıdır. Çeşitliliğin sağlanması için *limit* değeri 15 olarak belirlenmiştir. Diğer kontrol parametreleri ortalama iyi sonuç verdiği belirtilen [178] EvoSuite platformundaki varsayılan değerlerdir. Popülasyon sayısı 50, çaprazlama oranı 0.75, her bir operatör için (ekleme, silme, değiştirme) mutasyon oranı $1/3$ her bir test takımındaki maksimum test senaryosu sayısı 100 ve ceza faktörü w_i de 1 olarak alınmıştır. Gerçekleştirilen çalışmaların yaklaşık çalışma zamanı ihtiyacı 172.5 aydır.

¹<https://github.com/mauricioaniche/ck>

Tablo 3.1. Çaprazlama operatörlerinin etkisini incelemek için kullanılan sınıfların özellikleri

#	Sınıf Adı	Dal Sayısı	Satır Sayısı
1	com.aelitis.azureus.core.download.DownloadManagerEnhancer	96	13
2	com.lts.caloriecount.ui.budget.BudgetWin	24	118
3	com.lts.io.ArchiveScanner	44	86
4	com.lts.swing.table.dragndrop.test.RecordingEvent	18	95
5	com.lts.swing.thread.BlockThread	2	30
6	com.werken.saxpath.XPathLexer	458	408
7	corina.browser.Row	52	104
8	corina.formats.TRML	84	164
9	corina.graph.DensityPlot	14	49
10	corina.map.SiteListPanel	174	328
11	corina.prefs.components.BoolPrefComponent	6	17
12	de.huxhorn.lilith.data.access.AccessEvent	102	132
13	de.outstare.fortbattleplayer.gui.battlefield.BattlefieldCell	54	161
14	geo.google.mapping.AddressToUsAddressFunctor	28	40
15	httpanalyzer.ScreenInputFilter	4	14
16	jigl.image.utils.LocalDifferentialGeometry	196	506
17	lotus.core.phases.Phase	26	46
18	macaw.presentationLayer.CategoryStateEditor	10	43
19	ncs.Bessj	29	80
20	net.kencochrane.a4j.DAO.Cart	38	135
21	net.n3.nanoxml.XMLElement	310	661
22	net.sourceforge.squirrel_sql.fw.datasetviewer.ColumnDisplayDefinition	20	138
23	org.apache.commons.cli.CommandLine	45	87
24	org.apache.commons.codec.language.DoubleMetaphone	504	579
25	org.apache.commons.math.transform.FastFourierTransformer	135	290
26	org.bouncycastle.asn1.DERUTCTime	26	70
27	org.dom4j.util.PerThreadSingleton	4	23
28	org.fixsuite.message.view.ListView	62	235
29	org.gudy.azureus2.core3.torrentdownloader.impl.TorrentDownloaderManager	46	67
30	org.gudy.azureus2.core3.util.ShellUtilityFinder	6	12
31	org.jcvi.jillion.fasta.pos.PositionFastaRecord	12	35
32	org.jdom.Attribute	65	38
33	org.joda.time.format.DateTimeFormat	434	356
34	roops.core.bv32.arr.noex.IntArrayWithoutExceptions	43	64
35	roops.core.bv32.linear.noex.gods.LinearWithoutOverflow	93	223
36	roops.extended.bv32.arr.noex.IntArrayWithoutExceptionsWithArrayParameters	29	30
37	scs.Cookie	13	18
38	scs.DateParse	39	32
39	scs.Ordered4	29	11
40	scs.Stemmer	344	345
41	umd.cs.shop.JSListSubstitution	6	15
42	weka.filters.unsupervised.attribute.RandomProjection	138	299
43	weka.gui.beans.TrainTestSplitMaker	92	175

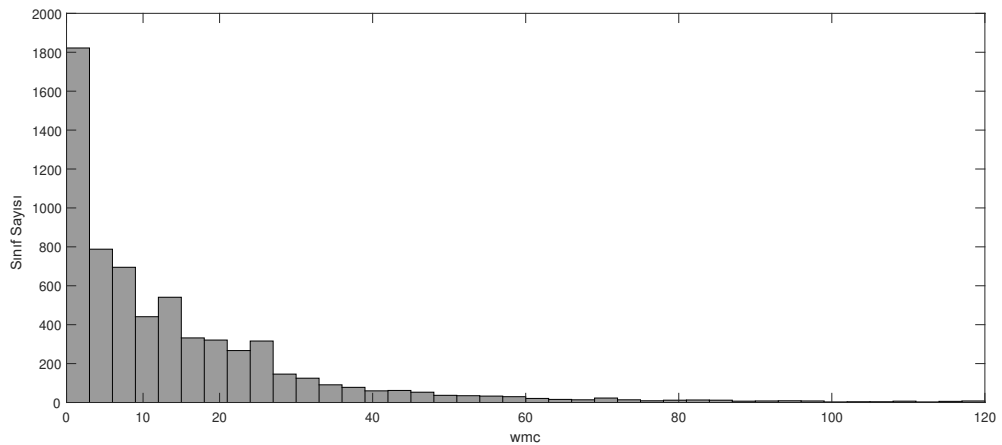
Tablo 3.2. Her bir seviyede sınıfların metriklerinin tanımlayıcı istatistik değerleri. Ortalama $\pm$ standart sapma değerleri ilk satırda, alt satırda parantez içinde ise ortanca değerleri verilmiştir.

	Seviye 1	Seviye 2	Seviye 3	Seviye 4	Seviye 5
Sınıf Sayısı	3424	1427	1130	493	95
Sınır(WMC)	$10 < WMC$	$10 \leq WMC < 20$	$20 \leq WMC < 40$	$40 \leq WMC < 120$	$120 \leq WMC$
WMC	3.6136 ± 2.3464 (2.0000)	13.9327 ± 2.7909 (14.0000)	26.6593 ± 5.1397 (26.0000)	61.6552 ± 20.2759 (55.0000)	204.2105 ± 127.8780 (156.0000)
CBO	2.0304 ± 2.2758 (1.0000)	5.6300 ± 3.8325 (6.0000)	8.9938 ± 5.5534 (9.0000)	12.4584 ± 10.8270 (10.0000)	23.6000 ± 27.1706 (15.0000)
DIT	2.2836 ± 1.3883 (2.0000)	1.4450 ± 1.1398 (1.0000)	1.6133 ± 1.4725 (1.0000)	2.5112 ± 3.8979 (1.0000)	3.0842 ± 4.2393 (1.0000)
NOC	0.2894 ± 2.9131 (0.0000)	0.5445 ± 1.6731 (0.0000)	1.6372 ± 35.1931 (0.0000)	0.2535 ± 1.0154 (0.0000)	0.1368 ± 0.4522 (0.0000)
RFC	3.0555 ± 4.5374 (1.0000)	9.5767 ± 10.6267 (7.0000)	17.0257 ± 16.8644 (14.0000)	46.1785 ± 27.9345 (44.0000)	89.8842 ± 65.2365 (78.0000)
LCOM	2.3280 ± 4.2910 (1.0000)	32.4036 ± 33.5559 (21.0000)	122.6248 ± 123.9377 (86.0000)	396.9026 ± 601.2618 (168.0000)	3431.2842 ± 12814.6667 (640.0000)
NOM	2.8700 ± 1.8541 (2.0000)	9.7631 ± 4.6007 (10.0000)	17.2920 ± 7.9229 (17.0000)	27.8276 ± 16.9484 (23.0000)	59.6316 ± 73.1385 (42.0000)
NOPM	2.6256 ± 1.8337 (2.0000)	8.9678 ± 5.1627 (9.0000)	15.5973 ± 9.0651 (15.0000)	22.9412 ± 18.1684 (18.0000)	42.9684 ± 72.2237 (24.0000)
NOSM	0.1533 ± 0.5701 (0.0000)	0.5417 ± 1.5287 (0.0000)	0.8186 ± 2.3723 (0.0000)	1.8824 ± 4.4000 (0.0000)	6.0842 ± 18.0600 (1.0000)
NOF	1.3414 ± 5.6199 (0.0000)	5.0855 ± 6.1546 (5.0000)	8.3655 ± 5.3546 (9.0000)	12.7465 ± 11.1619 (10.0000)	30.2211 ± 32.6738 (20.0000)
NOPF	0.4267 ± 5.2943 (0.0000)	0.8297 ± 5.0606 (0.0000)	0.7381 ± 1.9053 (0.0000)	1.3387 ± 5.4969 (0.0000)	2.3895 ± 8.2801 (0.0000)
NOSF	0.5514 ± 5.4261 (0.0000)	0.9026 ± 5.2908 (0.0000)	1.0265 ± 2.4938 (0.0000)	2.8540 ± 6.1838 (1.0000)	6.5053 ± 13.6902 (2.0000)
NOSI	0.5359 ± 1.4679 (0.0000)	1.8886 ± 7.4551 (0.0000)	3.5867 ± 6.8049 (0.0000)	10.9189 ± 13.0389 (7.0000)	28.3895 ± 34.9989 (16.0000)
LOC	50.4311 ± 28.0348 (40.0000)	125.9937 ± 46.5514 (122.0000)	224.3186 ± 77.3050 (224.0000)	425.7039 ± 189.7532 (387.0000)	1209.6737 ± 664.1804 (1091.0000)

Tablo 3.3. Tablo 3.2’de verilen metriklerin açıklamaları.

Kısaltma	Metrik
WMC	Ağırlıklandırılmış sınıf metodu metriği
CBO	Nesneler arası bağılılık
DIT	Kalıtım ağacı derinliği
NOC	Çocuk sayısı
RFC	Benzersiz metot çağrı sayısı
NOM	Metot sayısı
NOPM	Erişilebilir metot sayısı
NOSM	Statik metot sayısı
NOF	Alan sayısı
NOPF	Erişilebilir alan sayısı
NOSF	Statik alan sayısı
NOSI	Statik çağrı sayısı
LOC	Satır sayısı

Literatürdeki popüler algoritmalar olan MGA, SGA ve RS algoritmaları ile MABC algoritması kıyaslanıp istatistiksel analizlerle etkinliğini incelemek amacıyla toplam 34,949 daldan oluşan Panichella ve ark. (2018) [179] çalışmasındaki 180 sınıf üzerinde 2 dakika boyunca algoritmalar çalıştırılmıştır. Sınıfların dal sayısı [11, 7938] aralığında değişmektedir ve bu dal sayısının ortanca değeri 68’dir. Her bir algoritma [152, 165, 179] çalışmalarında olduğu gibi iki dakika boyunca çalışmıştır. Her bir istatistiksel analizlerin gerçekleştirilmesi için deneyler 20 sefer tekrarlanmıştır. İstatistiksel analiz olarak Wilcoxon testi yapılmış ve Vargha-Delaney $\hat{A}^{12}$ etki büyüklüğü değerleri verilmiştir.



Şekil 3.5. WMC değerlerinin histogram grafiği.

Son olarak, bir önceki deneyde verilen sınıflar üzerinde elde edilen popülasyonun

uygunluk değerlerinin standart sapması ve en iyi çözümlerin ortalamaları kullanılarak yakınsama ve çeşitlilik analizleri gerçekleştirilmiştir.

3.8.2. Çaprazlama Operatörünün İncelenmesi

Bu bölümde literatürde kullanılan çaprazlama tekniklerinin test senaryo üretimi için kullanılan ABC algoritması üzerindeki performansı incelenmiştir.

Elde edilen sonuçlara göre (Tablo 3.4), üç çaprazlama tekniği 17 sınıfta aynı sonucu elde etmiştir. SPR tekniği 12 sınıfta diğer iki tekniği geride bırakırken, 12 sınıfta SPF tekniği diğer iki tekniği geride bırakmıştır.

Sıralama sonuçları incelendiğinde, SP tekniğinin diğer iki teknikten başarısız olduğu görülmektedir. Bu sonuç ortanca ve ortalama değerleri incelendiğinde de görülmektedir. SPR tekniği SPF tekniğini sıralama sonuçlarına göre az farkla geçerken, ortanca ve ortalama değerlere göre ise az farkla geride kalmıştır.

Genel olarak değerlendirilecek olursa, SP tekniğinin ABC algoritması için uygun olmadığı görülmektedir. SPR ve SPF tekniklerinin ise birbirine benzer sonuçlar ürettiği görülmektedir. Ortalama, ortanca, standart sapma ve sıralama değerlerinin birbirine yakın olduğu da düşünülürse her iki tekniğin de problemin karakteristik özelliğine göre kullanılabileceği söylenebilmektedir. Sıralama değeri en düşük teknik SPR olduğu için çalışmanın devamında SPR kullanılmaktadır.

3.8.3. Arşiv Mekanizmasının Etkinliğinin İncelenmesi

Bu deneyde; arşiv mekanizmasının etkinliğini incelemek için ABC algoritmasının arşivli ve arşivsiz versiyonlarının elde etmiş olduğu dal kapsamı, satır kapsamı, çıkış kapsamı, zayıf mutasyon ve zaman sonuçları Tablo 3.5'te verilmiştir. Ortalama $\pm$ standart sapma değerleri ilk satırda, alt satırda parantez içinde ise ortanca değerleri sunulmuştur. Elde edilen sonuçlara göre, Seviye 1'de arşivsiz ABC algoritması; satır kapsamı, çıkış kapsamı ve zayıf mutasyon açısından ortanca değere göre daha iyi sonuç elde etmiştir. Diğer metrik ve seviyelerde ise arşivli ABC algoritması daha iyidir. Seviyenin, yani zorluğun artmasıyla birlikte aradaki fark daha fazla artmakta ve arşivli ABC algoritması daha

Tablo 3.4. Farklı çaprazlama tekniklerinin ABC algoritması üzerindeki etkileri. Ortalama $\pm$ standart sapma değerleri ilk satırda, alt satırda parantez içinde ise ortanca değerleri verilmiştir.

#	SPR	SP	SPF
1	0.0275 $\pm$ 0.0000 (0.0275) 1	0.0275 $\pm$ 0.0000 (0.0275) 1	0.0275 $\pm$ 0.0000 (0.0275) 1
2	0.1250 $\pm$ 0.0000 (0.1250) 1	0.1250 $\pm$ 0.0000 (0.1250) 1	0.1250 $\pm$ 0.0000 (0.1250) 1
3	0.6556 $\pm$ 0.1536 (0.6667) 2	0.5919 $\pm$ 0.1230 (0.6000) 3	0.6733 $\pm$ 0.1486 (0.6778) 1
4	0.9911 $\pm$ 0.0150 (1.0000) 3	0.9956 $\pm$ 0.0115 (1.0000) 2	0.9978 $\pm$ 0.0085 (1.0000) 1
5	1.0000 $\pm$ 0.0000 (1.0000) 1	1.0000 $\pm$ 0.0000 (1.0000) 1	0.9963 $\pm$ 0.0203 (1.0000) 2
6	0.6846 $\pm$ 0.0259 (0.6839) 2	0.6213 $\pm$ 0.0240 (0.6188) 3	0.6959 $\pm$ 0.0300 (0.7004) 1
7	0.4793 $\pm$ 0.0900 (0.4138) 2	0.4448 $\pm$ 0.0563 (0.4138) 3	0.4925 $\pm$ 0.0872 (0.4655) 1
8	0.3303 $\pm$ 0.0572 (0.3258) 2	0.3086 $\pm$ 0.0230 (0.3146) 3	0.3315 $\pm$ 0.0691 (0.3258) 1
9	0.1765 $\pm$ 0.0000 (0.1765) 1	0.1765 $\pm$ 0.0000 (0.1765) 1	0.1765 $\pm$ 0.0000 (0.1765) 1
10	0.0051 $\pm$ 0.0000 (0.0051) 1	0.0051 $\pm$ 0.0000 (0.0051) 1	0.0051 $\pm$ 0.0000 (0.0051) 1
11	0.0000 $\pm$ 0.0000 (0.0000) 1	0.0000 $\pm$ 0.0000 (0.0000) 1	0.0000 $\pm$ 0.0000 (0.0000) 1
12	0.9478 $\pm$ 0.0185 (0.9478) 1	0.8142 $\pm$ 0.0368 (0.8134) 3	0.9448 $\pm$ 0.0207 (0.9403) 2
13	0.7540 $\pm$ 0.0523 (0.7535) 1	0.7164 $\pm$ 0.0833 (0.7324) 3	0.7455 $\pm$ 0.0531 (0.7465) 2
14	0.2000 $\pm$ 0.0000 (0.2000) 1	0.2000 $\pm$ 0.0000 (0.2000) 1	0.2000 $\pm$ 0.0000 (0.2000) 1
15	0.9667 $\pm$ 0.0758 (1.0000) 1	0.9600 $\pm$ 0.0814 (1.0000) 2	0.9533 $\pm$ 0.0860 (1.0000) 3
16	0.4367 $\pm$ 0.0451 (0.4293) 2	0.3838 $\pm$ 0.0469 (0.3864) 3	0.4414 $\pm$ 0.0598 (0.4495) 1
17	0.5000 $\pm$ 0.0000 (0.5000) 1	0.5000 $\pm$ 0.0000 (0.5000) 1	0.5000 $\pm$ 0.0000 (0.5000) 1
18	0.0833 $\pm$ 0.0000 (0.0833) 1	0.0833 $\pm$ 0.0000 (0.0833) 1	0.0833 $\pm$ 0.0000 (0.0833) 1
19	0.9655 $\pm$ 0.0000 (0.9655) 1	0.9655 $\pm$ 0.0000 (0.9655) 1	0.9655 $\pm$ 0.0000 (0.9655) 1
20	0.2308 $\pm$ 0.0000 (0.2308) 1	0.2308 $\pm$ 0.0000 (0.2308) 1	0.2308 $\pm$ 0.0000 (0.2308) 1
21	0.9448 $\pm$ 0.0270 (0.9458) 2	0.8056 $\pm$ 0.0487 (0.8133) 3	0.9404 $\pm$ 0.0421 (0.9518) 1
22	0.9000 $\pm$ 0.0217 (0.8980) 2	0.8871 $\pm$ 0.0256 (0.8980) 3	0.9054 $\pm$ 0.0197 (0.9184) 1
23	0.9855 $\pm$ 0.0081 (0.9811) 1	0.9704 $\pm$ 0.0342 (0.9811) 3	0.9849 $\pm$ 0.0077 (0.9811) 2
24	0.7214 $\pm$ 0.0216 (0.7239) 1	0.6726 $\pm$ 0.0212 (0.6707) 3	0.7212 $\pm$ 0.0202 (0.7229) 2
25	0.6304 $\pm$ 0.0023 (0.6296) 2	0.6304 $\pm$ 0.0023 (0.6296) 2	0.6314 $\pm$ 0.0032 (0.6296) 1
26	0.9990 $\pm$ 0.0054 (1.0000) 1	0.9980 $\pm$ 0.0075 (1.0000) 2	0.9990 $\pm$ 0.0054 (1.0000) 1
27	1.0000 $\pm$ 0.0000 (1.0000) 1	1.0000 $\pm$ 0.0000 (1.0000) 1	1.0000 $\pm$ 0.0000 (1.0000) 1
28	0.1029 $\pm$ 0.0000 (0.1029) 1	0.1029 $\pm$ 0.0000 (0.1029) 1	0.1029 $\pm$ 0.0000 (0.1029) 1
29	0.8895 $\pm$ 0.0975 (0.9314) 1	0.8791 $\pm$ 0.1128 (0.9020) 3	0.8902 $\pm$ 0.0834 (0.9020) 2
30	0.8889 $\pm$ 0.0000 (0.8889) 1	0.8889 $\pm$ 0.0000 (0.8889) 1	0.8889 $\pm$ 0.0000 (0.8889) 1
31	1.0000 $\pm$ 0.0000 (1.0000) 1	1.0000 $\pm$ 0.0000 (1.0000) 1	1.0000 $\pm$ 0.0000 (1.0000) 1
32	0.8959 $\pm$ 0.0241 (0.8923) 2	0.7933 $\pm$ 0.0255 (0.7846) 3	0.9051 $\pm$ 0.0274 (0.9077) 1
33	0.6492 $\pm$ 0.0158 (0.6463) 2	0.5691 $\pm$ 0.0727 (0.5922) 3	0.6563 $\pm$ 0.0417 (0.6613) 1
34	0.9302 $\pm$ 0.0000 (0.9302) 1	0.9302 $\pm$ 0.0000 (0.9302) 1	0.9302 $\pm$ 0.0000 (0.9302) 1
35	0.9624 $\pm$ 0.0254 (0.9785) 1	0.8513 $\pm$ 0.0204 (0.8495) 3	0.9534 $\pm$ 0.0287 (0.9570) 2
36	0.9989 $\pm$ 0.0063 (1.0000) 1	0.9966 $\pm$ 0.0139 (1.0000) 2	0.9920 $\pm$ 0.0196 (1.0000) 3
37	0.9769 $\pm$ 0.0359 (1.0000) 1	0.9513 $\pm$ 0.0514 (0.9231) 3	0.9718 $\pm$ 0.0428 (1.0000) 2
38	1.0000 $\pm$ 0.0000 (1.0000) 1	1.0000 $\pm$ 0.0000 (1.0000) 1	1.0000 $\pm$ 0.0000 (1.0000) 1
39	0.9034 $\pm$ 0.0698 (0.9310) 2	0.8862 $\pm$ 0.0801 (0.9310) 3	0.9310 $\pm$ 0.0373 (0.9310) 1
40	0.4755 $\pm$ 0.0817 (0.4753) 1	0.3920 $\pm$ 0.0545 (0.3852) 3	0.4712 $\pm$ 0.0657 (0.4578) 2
41	1.0000 $\pm$ 0.0000 (1.0000) 1	1.0000 $\pm$ 0.0000 (1.0000) 1	1.0000 $\pm$ 0.0000 (1.0000) 1
42	0.3812 $\pm$ 0.0179 (0.3797) 1	0.3489 $\pm$ 0.0184 (0.3481) 3	0.3762 $\pm$ 0.0258 (0.3797) 2
43	0.4280 $\pm$ 0.0273 (0.4340) 1	0.3752 $\pm$ 0.0286 (0.3774) 3	0.4324 $\pm$ 0.0337 (0.4292) 2
Tanımlayıcı İstatistik	0.6564 $\pm$ 0.3481 (0.7606)	0.6298 $\pm$ 0.3429 (0.7324)	0.6574 $\pm$ 0.3478 (0.7746)
TOPLAM	56	88	58

üstün hale gelmektedir. Bu da arşiv mekanizmasının, özellikle kompleks problemlerde, algoritmanın performansını artırdığını göstermektedir. Çalışma zamanı incelendiğinde ise arşiv mekanizmasının algoritmanın çalışma zamanını da düşürdüğü görülmektedir çünkü maksimum kapsama miktarına daha hızlı erişmesini sağlamaktadır. Bu nedenle arşiv mekanizmasının kullanılmasının algoritma üzerinde olumlu etkileri olduğu görülmüştür.

3.8.4. Algoritmaların Problem Zorluklarına Göre İncelenmesi

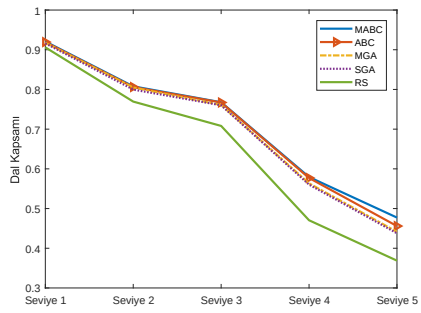
Önceki bölümde arşiv mekanizmasının ABC algoritmasının performansına önemli ölçüde katkıda bulunduğu gösterilmiştir. Bu bölümde ise arşiv tabanlı ABC ve MABC algoritmalarının performansı, literatürde sıkça kullanılan MGA, SGA ve RS algoritmaları ile kıyaslanmıştır. Bu bölümde de bir önceki bölümde kullanılan problem seti üzerinde sonuçlar alınmış ve dal kapsamı, satır kapsamı, çıkış kapsamı, zayıf mutasyon ve zaman açısından sonuçlar seviyelere göre Tablo 3.6'da ortalama $\pm$ standart sapma değerleri ilk satırda, alt satırda parantez içinde ise ortanca değerleri olarak verilmiştir..

Dal kapsamı açısından incelendiğinde, Seviye 1 ve 2'de ortanca değerleri tüm algoritmalarda eşit iken ortalama ve standart sapmaları ise birbirine yakındır. Seviye arttıkça, aralarındaki fark da artmaktadır. Bütün seviyelerde ortalama değere göre MABC en iyi algoritma olurken ardından sırasıyla ABC, MGA, SGA ve RS gelmektedir. Satır kapsamı sonuçları da, Seviye 4 hariç dal kapsamına benzemektedir. Seviye 4'te en başarılı algoritma ABC olurken ardından MABC gelmektedir. MGA ise SGA'nın gerisinde kalmış, RS'ı ise geçmiştir. Çıkış kapsamı incelendiğinde, MABC en fazla kapsama miktarına erişen algoritmadır. ABC ise MABC algoritmasından sonra ikincidir. RS ise en başarısız algoritma olmuştur. SGA Seviye 1, 3 ve 4'te MGA'yı geride bırakırken, MGA da SGA'yı Seviye 2 ve 5'te geçmiştir. Ortalama zayıf mutasyon sonuçlarına göre, MABC diğer algoritmaları geride bırakmıştır. ABC ise ikinci sırada bulunurken RS en başarısız algoritma olmuştur. Seviye 1,2 ve 4'te MGA, SGA algoritmasını geride bırakırken, SGA da MGA algoritmasını Seviye 3 ve 5'te geçmiştir.

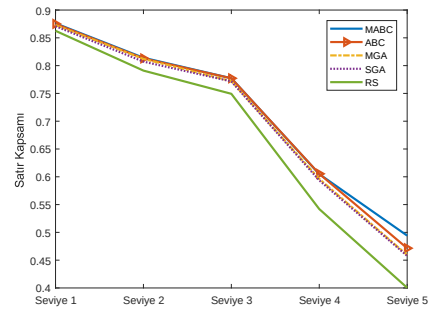
Tablo 3.5. ABC algoritmasının arşivli ve arşivsiz versiyonlarının karşılaştırılması. Ortalama $\pm$ standart sapma değerleri ilk satırda, alt satırda parantez içinde ise ortanca değerleri verilmiştir.

Kapsam	Arşiv	Zorluk Seviyesi				
		Seviye 1	Seviye 2	Seviye 3	Seviye 4	Seviye 5
Dal	Hayır	0.9189 $\pm$ 0.2172 (1.0000)	0.8033 $\pm$ 0.3135 (1.0000)	0.7628 $\pm$ 0.3360 (0.9677)	0.5643 $\pm$ 0.3671 (0.6143)	0.4344 $\pm$ 0.3736 (0.3491)
	Evet	0.9193 $\pm$ 0.2179 (1.0000)	0.8061 $\pm$ 0.3132 (1.0000)	0.7672 $\pm$ 0.3372 (0.9733)	0.5770 $\pm$ 0.3706 (0.6374)	0.4559 $\pm$ 0.3822 (0.3864)
Satır	Hayır	0.8760 $\pm$ 0.2592 (1.0000)	0.8118 $\pm$ 0.3054 (1.0000)	0.7748 $\pm$ 0.3271 (0.9672)	0.5965 $\pm$ 0.3566 (0.6765)	0.4548 $\pm$ 0.3621 (0.3867)
	Evet	0.8751 $\pm$ 0.2607 (1.0000)	0.8130 $\pm$ 0.3061 (1.0000)	0.7772 $\pm$ 0.3285 (0.9756)	0.6054 $\pm$ 0.3573 (0.6845)	0.4715 $\pm$ 0.3676 (0.4392)
Çıkış	Hayır	0.8273 $\pm$ 0.3003 (1.0000)	0.7273 $\pm$ 0.3323 (0.8947)	0.6896 $\pm$ 0.3370 (0.8000)	0.5230 $\pm$ 0.3434 (0.5143)	0.4184 $\pm$ 0.3405 (0.3625)
	Evet	0.8251 $\pm$ 0.3003 (1.0000)	0.7343 $\pm$ 0.3336 (0.9130)	0.7007 $\pm$ 0.3392 (0.8000)	0.5331 $\pm$ 0.3490 (0.5299)	0.4480 $\pm$ 0.3424 (0.4124)
Zayıf	Hayır	0.8870 $\pm$ 0.2823 (1.0000)	0.7814 $\pm$ 0.3464 (0.9839)	0.7432 $\pm$ 0.3601 (0.9522)	0.5641 $\pm$ 0.3772 (0.6496)	0.4456 $\pm$ 0.3744 (0.3469)
	Evet	0.8866 $\pm$ 0.2837 (1.0000)	0.7825 $\pm$ 0.3487 (0.9888)	0.7447 $\pm$ 0.3616 (0.9547)	0.5730 $\pm$ 0.3782 (0.6700)	0.4557 $\pm$ 0.3771 (0.4279)
Zaman	Hayır	118.8759 $\pm$ 737.8715 (29.0115)	197.9447 $\pm$ 868.7278 (65.3980)	335.9842 $\pm$ 965.3724 (140.2640)	901.0043 $\pm$ 1403.4981 (480.4020)	3070.4904 $\pm$ 3364.4752 (1875.6700)
	Evet	75.3557 $\pm$ 590.5981 (8.3930)	137.7949 $\pm$ 752.0249 (32.0530)	221.3067 $\pm$ 1000.9601 (50.0665)	406.8962 $\pm$ 997.1988 (175.4880)	1875.1879 $\pm$ 2880.1034 (730.7750)

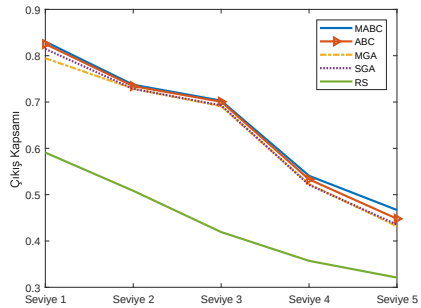
Çalışma zamanı ele alındığında en hızlı çalışan algoritma RS'dir. MABC ise Seviye 1'de ikinci en hızlı algoritma olmuştur ancak diğer seviyelerde diğer algoritmalarından daha fazla hesaplama zamanına ihtiyaç duymaktadır. MGA ve SGA algoritmaları benzer zamana ihtiyaç duyarken, ABC MABC'den sonra ikinci en yavaş algoritma olmuştur. Seviyelere göre ortalama kapsama miktarı ve geçen zamanlar Şekil 3.6'da verilmiştir. Şekil 3.6'ya göre seviye arttıkça kapsama miktarı düşmektedir. Bu da sınıfların doğru bir şekilde problem zorluğuna göre bölündüğünü göstermektedir.



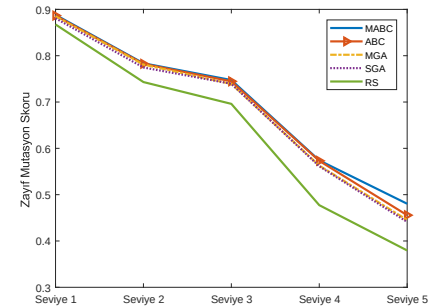
(a) Dal Kapsamı



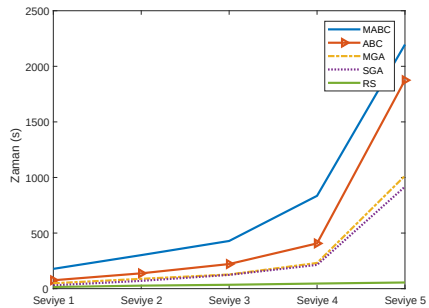
(b) Satır Kapsamı



(c) Çıkış Kapsamı



(d) Zayıf Mutasyon



(e) Zaman

Şekil 3.6. Algoritmalar tarafından üretilen kapsama miktarlarının her bir seviyeye göre sonuçları.

Tablo 3.6. MABC ve ABC algoritmalarının literatürdeki popüler algoritmalar ile kıyaslanması. Ortalama $\pm$ standart sapma değerleri ilk satırda, alt satırda parantez içinde ise ortalama değerleri verilmiştir.

Kapsama	Algoritma	Zorluk Seviyesi				
		Seviye 1	Seviye 2	Seviye 3	Seviye 4	Seviye 5
Dal Kapsamı	RS	0.9060 $\pm$ 0.2332 (1.0000)	0.7694 $\pm$ 0.3245 (1.0000)	0.7081 $\pm$ 0.3414 (0.8524)	0.4705 $\pm$ 0.3377 (0.4560)	0.3688 $\pm$ 0.3533 (0.2444)
	MGA	0.9176 $\pm$ 0.2192 (1.0000)	0.8049 $\pm$ 0.3141 (1.0000)	0.7608 $\pm$ 0.3395 (0.9677)	0.5635 $\pm$ 0.3667 (0.6118)	0.4414 $\pm$ 0.3733 (0.3550)
	SGA	0.9162 $\pm$ 0.2215 (1.0000)	0.7996 $\pm$ 0.3174 (1.0000)	0.7601 $\pm$ 0.3408 (0.9655)	0.5603 $\pm$ 0.3682 (0.6053)	0.4377 $\pm$ 0.3750 (0.3350)
	ABC	0.9193 $\pm$ 0.2179 (1.0000)	0.8061 $\pm$ 0.3132 (1.0000)	0.7672 $\pm$ 0.3372 (0.9783)	0.5770 $\pm$ 0.3706 (0.6374)	0.4559 $\pm$ 0.3822 (0.3864)
	MABC	0.9207 $\pm$ 0.2159 (1.0000)	0.8078 $\pm$ 0.3134 (1.0000)	0.7677 $\pm$ 0.3402 (0.9730)	0.5794 $\pm$ 0.3752 (0.6615)	0.4773 $\pm$ 0.3860 (0.4435)
	RS	0.8626 $\pm$ 0.2713 (1.0000)	0.7910 $\pm$ 0.3155 (1.0000)	0.7493 $\pm$ 0.3380 (0.9545)	0.5420 $\pm$ 0.3513 (0.5443)	0.4003 $\pm$ 0.3503 (0.2763)
Süre Kapsamı	MGA	0.8728 $\pm$ 0.2628 (1.0000)	0.8121 $\pm$ 0.3067 (1.0000)	0.7721 $\pm$ 0.3309 (0.9697)	0.5963 $\pm$ 0.3549 (0.6560)	0.4593 $\pm$ 0.3634 (0.3847)
	SGA	0.8709 $\pm$ 0.2630 (1.0000)	0.8072 $\pm$ 0.3098 (1.0000)	0.7716 $\pm$ 0.3322 (0.9685)	0.5935 $\pm$ 0.3569 (0.6726)	0.4577 $\pm$ 0.3635 (0.3745)
	ABC	0.8751 $\pm$ 0.2607 (1.0000)	0.8130 $\pm$ 0.3061 (1.0000)	0.7772 $\pm$ 0.3285 (0.9756)	0.6054 $\pm$ 0.3573 (0.6845)	0.4715 $\pm$ 0.3676 (0.4392)
	MABC	0.8761 $\pm$ 0.2592 (1.0000)	0.8142 $\pm$ 0.3056 (1.0000)	0.7773 $\pm$ 0.3310 (0.9756)	0.6049 $\pm$ 0.3604 (0.6821)	0.4942 $\pm$ 0.3748 (0.4556)
	RS	0.5908 $\pm$ 0.2924 (0.5000)	0.5084 $\pm$ 0.2848 (0.5000)	0.4193 $\pm$ 0.2552 (0.4321)	0.3571 $\pm$ 0.2975 (0.3333)	0.3210 $\pm$ 0.3338 (0.2000)
	MGA	0.7945 $\pm$ 0.3074 (1.0000)	0.7289 $\pm$ 0.3320 (0.9000)	0.6910 $\pm$ 0.3400 (0.7857)	0.5228 $\pm$ 0.3480 (0.5000)	0.4317 $\pm$ 0.3326 (0.4000)
Çıkış Kapsamı	SGA	0.8146 $\pm$ 0.3052 (1.0000)	0.7284 $\pm$ 0.3354 (0.9000)	0.6928 $\pm$ 0.3418 (0.7977)	0.5212 $\pm$ 0.3461 (0.5063)	0.4352 $\pm$ 0.3403 (0.3913)
	ABC	0.8251 $\pm$ 0.3003 (1.0000)	0.7343 $\pm$ 0.3336 (0.9130)	0.7007 $\pm$ 0.3392 (0.8000)	0.5331 $\pm$ 0.3490 (0.5299)	0.4480 $\pm$ 0.3424 (0.4124)
	MABC	0.8304 $\pm$ 0.2992 (1.0000)	0.7373 $\pm$ 0.3332 (0.9167)	0.7030 $\pm$ 0.3420 (0.8153)	0.5407 $\pm$ 0.3533 (0.5455)	0.4669 $\pm$ 0.3484 (0.4629)
	RS	0.8674 $\pm$ 0.3004 (1.0000)	0.7431 $\pm$ 0.3618 (0.9595)	0.6960 $\pm$ 0.3681 (0.8824)	0.4773 $\pm$ 0.3506 (0.4771)	0.3795 $\pm$ 0.3548 (0.2555)
	MGA	0.8850 $\pm$ 0.2855 (1.0000)	0.7809 $\pm$ 0.3479 (0.9880)	0.7391 $\pm$ 0.3645 (0.9487)	0.5627 $\pm$ 0.3742 (0.6235)	0.4453 $\pm$ 0.3757 (0.3410)
	SGA	0.8808 $\pm$ 0.2905 (1.0000)	0.7746 $\pm$ 0.3525 (0.9873)	0.7394 $\pm$ 0.3647 (0.9512)	0.5613 $\pm$ 0.3759 (0.6204)	0.4411 $\pm$ 0.3723 (0.3582)
Zayıf Mutasyon	ABC	0.8866 $\pm$ 0.2837 (1.0000)	0.7825 $\pm$ 0.3487 (0.9888)	0.7447 $\pm$ 0.3616 (0.9547)	0.5730 $\pm$ 0.3782 (0.6700)	0.4557 $\pm$ 0.3771 (0.4279)
	MABC	0.8882 $\pm$ 0.2814 (1.0000)	0.7837 $\pm$ 0.3481 (0.9888)	0.7474 $\pm$ 0.3630 (0.9556)	0.5740 $\pm$ 0.3816 (0.6655)	0.4804 $\pm$ 0.3826 (0.4642)
	RS	14.8120 $\pm$ 149.6039 (3.1230)	27.0752 $\pm$ 181.1192 (8.9610)	34.8004 $\pm$ 166.4685 (12.5095)	45.9340 $\pm$ 137.4421 (20.1290)	55.3932 $\pm$ 88.2118 (38.6820)
	MGA	48.3483 $\pm$ 483.9627 (9.1100)	88.3022 $\pm$ 636.4947 (19.6820)	127.3994 $\pm$ 599.4217 (33.7435)	230.9297 $\pm$ 606.7602 (98.2690)	1015.3165 $\pm$ 2059.7354 (309.3730)
	SGA	29.8802 $\pm$ 243.7913 (7.6145)	70.0246 $\pm$ 425.7181 (18.3200)	123.7836 $\pm$ 545.3861 (33.2555)	213.6786 $\pm$ 503.4690 (104.8180)	917.4600 $\pm$ 1883.5110 (280.3800)
	ABC	75.3557 $\pm$ 590.5981 (8.3930)	137.7949 $\pm$ 752.0249 (32.0530)	221.3067 $\pm$ 1000.9601 (50.0665)	406.8962 $\pm$ 997.1988 (175.4880)	1875.1879 $\pm$ 2880.1034 (730.7750)
Zaman	MABC	178.0313 $\pm$ 957.4972 (9.0830)	301.6160 $\pm$ 1100.7878 (78.0240)	428.8797 $\pm$ 1384.4179 (119.4315)	834.9361 $\pm$ 1618.5887 (338.8900)	2196.0750 $\pm$ 2882.8643 (1190.3830)

3.8.5. İstatistiksel Analiz

İstatistiksel analizlerin gerçekleştirilebilmesi için MABC, ABC, MGA, SGA ve RS algoritmaları 180 sınıf üzerinde 20 sefer çalıştırılmış ve projelere göre elde edilen ortalama dal kapsamı sonuçları Tablo 3.7’de verilmiştir. Tablonun son satırlarında tanımlayıcı istatistik ve sıralama sonuçları verilmiştir. Ortanca değerlere göre MABC ve ABC algoritmaları en iyi sonuca erişen algoritmalar olmuştur. SGA ikinci sırada bulunurken MGA ise üçüncü sıradadır. En kötü performansa sahip algoritma ise RS’dir. Ortalama değere göre MABC en iyi algoritma olurken ardından ABC, MGA, SGA ve RS gelmektedir. Sınıf tabanlı sıralamaya göre ise MABC’nin ortalaması 0.5159 olurken MGA 0.5125 ortalama sahiptir. ABC algoritması ise 0.5108 ortalama ile üçüncü sırada bulunurken SGA 0.5053 ile RS’den sonra en kötü ikinci algoritma olmuştur. RS ise 0.4914 ortalama sahiptir. Aynı süre içerisinde MABC algoritmasının daha yavaş çalışması ve daha az iterasyon gerçekleştirmesine rağmen daha yüksek kapsama miktarına erişmiştir. Her bir sınıfa ait tablolar ise EK-1’de verilmiştir. Her bir sınıfa ait ortalama dal kapsamı değerleri ile birlikte tanımlayıcı istatistik ve sıralama değerleri Tablo EK-1.2’de, bu sonuçlara ait Wilcoxon test sonuçları ise Tablo EK-1.1’de verilmiştir. Bu tablolara göre en iyi ortalama ve sıralama değerlerine sahip algoritma MABC olurken ardından sırasıyla, MGA, ABC, SGA ve RS gelmektedir.

Tablo 3.7. Proje bazlı elde edilen sonuçların Dal Kapsamı metriğine göre 20 bağımsız koşmanın ortalaması.

PID	Proje Adı	Sınıf Sayısı	MABC	ABC	MGA	SGA	RS
1	tullibee	2	0.9938	0.9975	0.9986	0.9962	0.7837
2	a4j	2	0.2742	0.2902	0.2772	0.28	0.2619
4	rif	1	1	1	1	1	0.9098
5	templateit	2	0.6026	0.6749	0.6441	0.656	0.5595
7	sfmis	2	0.7261	0.6999	0.6994	0.7101	0.7047
10	water-simulator	1	0.1179	0.1145	0.1188	0.1183	0.1199
11	imsmart	1	1	1	1	1	1
12	dsachat	2	0.2373	0.2171	0.2329	0.2154	0.2485
13	jdbacl	2	0.7241	0.6536	0.6739	0.6727	0.719
14	omjstate	1	1	1	1	0.9981	0.9017
15	beanbin	2	0.7442	0.7044	0.7158	0.7208	0.7401
17	inspirento	2	0.8406	0.8366	0.8348	0.8401	0.7851
18	jsecurity	3	0.6172	0.6008	0.6079	0.6088	0.5916

Tablo 3.7. Proje bazlı elde edilen sonuçların Dal Kapsamı metriğine göre 20 bağımsız koşmanın ortalaması.

PID	Proje Adı	Sınıf Sayısı	MABC	ABC	MGA	SGA	RS
19	jmca	4	0.4437	0.3815	0.3899	0.3926	0.405
21	geo-google	2	0.8117	0.8462	0.8509	0.8383	0.7595
22	byuic	4	0.4148	0.4045	0.4124	0.4081	0.447
24	saxpath	2	0.913	0.8448	0.8516	0.8599	0.8413
26	jipa	2	0.8006	0.7704	0.7784	0.7946	0.8282
27	gangup	2	0.2544	0.2544	0.2486	0.2489	0.2325
29	apbsmem	2	0.4865	0.5135	0.4565	0.4744	0.4673
31	xisemele	1	0.5556	0.5556	0.5574	0.5556	0.5068
32	httpanalyzer	2	0.4093	0.4093	0.3773	0.4093	0.4093
33	javaviewcontrol	3	0.5236	0.5248	0.5226	0.5324	0.4798
35	corina	3	0.0908	0.0974	0.0903	0.09	0.0897
36	schemaspy	2	0.2018	0.2823	0.302	0.2158	0.1476
37	petsoar	2	0.7553	0.7553	0.7487	0.7553	0.7483
38	javabullboard	2	0.6379	0.6297	0.6447	0.6428	0.5588
39	diffi	2	0.9037	0.9194	0.9089	0.9121	0.8844
40	glengineer	2	0.7846	0.803	0.7993	0.8067	0.7235
41	follow	2	0.6161	0.5829	0.5817	0.5454	0.5346
42	asphodel	1	0.0238	0.0238	0.0238	0.0238	0.0238
43	lilith	4	0.6402	0.6497	0.6428	0.6532	0.5541
44	summa	2	0.476	0.5415	0.5186	0.5021	0.4241
45	lotus	2	0.4583	0.4583	0.4605	0.4583	0.4583
46	nutzenportfolio	2	0.1914	0.2188	0.2155	0.2188	0.1912
47	dvd-homevideo	3	0.092	0.0924	0.0931	0.0909	0.092
49	diebierse	2	0.659	0.6695	0.6537	0.6614	0.6337
50	biff	1	0.1087	0.1067	0.1168	0.0202	0.0391
51	jiprof	4	0.5277	0.5271	0.5412	0.5244	0.4985
52	lagoon	2	0.1591	0.149	0.1521	0.1491	0.1637
54	db-everywhere	2	0.3725	0.3462	0.3458	0.3587	0.3993
55	lavalamp	1	1	1	1	1	1
56	jhandballmoves	2	0.1967	0.1926	0.2007	0.1926	0.1913
57	hft-bombberman	2	0.411	0.4952	0.4518	0.4471	0.3325
58	fps370	2	0.0105	0.0104	0.0106	0.0105	0.0103
59	mygrid	2	0.8214	0.8214	0.8214	0.8214	0.7887
60	sugar	2	0.8412	0.8324	0.8325	0.8255	0.856
61	noen	2	0.7788	0.8165	0.7997	0.7773	0.6844
62	dom4j	2	0.4011	0.3255	0.3273	0.343	0.437
63	objectexplorer	2	0.4424	0.419	0.4635	0.4615	0.3968
64	jtaiogui	2	0.556	0.5549	0.5246	0.2725	0.4784
65	gsftp	2	0.2367	0.2032	0.2222	0.2172	0.2854
66	openjms	2	0.6424	0.6437	0.6541	0.6442	0.5871
67	gae-app-manager	1	0.1489	0.1489	0.1489	0.1489	0.1489
68	biblestudy	2	0.7969	0.7969	0.7911	0.8021	0.7655
69	lhamacaw	2	0.2148	0.2694	0.2598	0.2115	0.1587
70	echodep	2	0.0104	0.0104	0.0104	0.0104	0.0104

Tablo 3.7. Proje bazlı elde edilen sonuçların Dal Kapsamı metriğine göre 20 bağımsız koşmanın ortalaması.

PID	Proje Adı	Sınıf Sayısı	MABC	ABC	MGA	SGA	RS
71	ext4j	2	0.9629	0.9589	0.9642	0.9638	0.895
72	battlecry	2	0.0878	0.0762	0.058	0.0589	0.1483
73	fiml	2	0.0821	0.0821	0.08	0.08	0.08
74	fixsuite	2	0.2741	0.2613	0.2574	0.251	0.2718
75	openhre	2	0.9969	0.9954	0.9964	0.9972	0.9415
77	io-project	1	0.8545	0.925	0.922	0.8939	0.7833
78	caloriecount	3	0.8843	0.8553	0.8602	0.8557	0.8667
79	twfbplayer	2	0.8012	0.8033	0.802	0.7812	0.7627
80	wheelwebtool	4	0.7325	0.7375	0.73	0.732	0.7029
81	javathena	4	0.5157	0.5046	0.5202	0.5123	0.5238
82	ipcalculator	2	0.7097	0.6592	0.6773	0.6548	0.7016
83	xbus	2	0.1762	0.1762	0.1769	0.1762	0.1741
84	ifx-framework	1	0.4306	0.4264	0.4306	0.4306	0.4306
85	shop	4	0.5423	0.5404	0.5509	0.528	0.5781
86	at-robots2-j	2	0.6167	0.5423	0.5529	0.5278	0.6263
87	jaw-br	1	0.1923	0.1923	0.1923	0.1923	0.1923
88	jopenchart	2	0.4731	0.431	0.4305	0.4366	0.4444
89	jiggler	1	0.941	0.9424	0.947	0.9318	0.8621
100	jgaap	1	0.8261	0.8152	0.8164	0.813	0.8261
101	netweaver	2	0.9274	0.9292	0.9287	0.9292	0.7062
102	squirrel-sql	2	0.4419	0.488	0.4924	0.4847	0.4136
103	sweethome3d	4	0.0771	0.0716	0.0716	0.0729	0.0782
104	vuze	2	0.0204	0.0203	0.0202	0.0201	0.0147
105	freemind	2	0.5401	0.528	0.5158	0.5247	0.5117
106	checkstyle	1	0.1081	0.1081	0.1081	0.1081	0.1081
107	weka	4	0.5186	0.473	0.4568	0.4798	0.5408
108	liferay	1	1	1	1	1	0.9791
109	pdfsam	2	0.396	0.3899	0.396	0.396	0.3812
110	firebird	3	0.9087	0.8622	0.9105	0.895	0.8878
	1. Çeyrek		0.2416	0.2633	0.258	0.2263	0.2518
	Ortanca		0.5339	0.5342	0.5236	0.5246	0.5093
	3. Çeyrek		0.7997	0.8032	0.7996	0.8002	0.7567
	Ortalama		0.5318	0.5289	0.5287	0.5218	0.505
Sıralamaların Ortalaması			2.5	2.9593	2.7616	3.0814	3.6977

Tablo 3.8, dal kapsamı sonuçlarında gerçekleştirilen Wilcoxon testine göre ($p - \text{degeri} \leq 0.05$) bir algoritmanın diğer algoritmadan kaç sınıfta daha iyi olduğunu göstermektedir. Son satırda ise toplam kaç sınıfta iyi veya kötü olduğu verilmiştir. Wilcoxon testi Holm-Bonferroni düzeltme yöntemi ile birlikte kullanılmıştır. Tablo 3.8'deki sonuçlara göre MABC algoritması 166 sınıfta diğer algoritmalarından daha iyidir ve 55 sınıfta da

Tablo 3.8. Sütundaki bir algoritmanın satırdaki algoritmaya Dal Kapsamı sonuçlarına uygulanan Wilcoxon testi ($p - degeri \leq 0.05$) sonucunda üstünlük sağladığı sınıf sayısı İİ: İstatistiksel olarak daha iyi sınıf sayısı, İK: İstatistiksel olarak daha kötü sınıf sayısı.

	MABC		ABC		MGA		SGA		RS	
	İK	İİ	İK	İİ	İK	İİ	İK	İİ	İK	İİ
MABC	-	-	34	13	32	14	34	10	66	18
ABC	13	34	-	-	3	6	9	4	67	31
MGA	14	32	6	3	-	-	7	5	67	34
SGA	10	34	4	9	5	7	-	-	60	35
RS	18	66	31	67	34	67	35	60	-	-
TOPLAM	55	166	75	92	74	94	85	79	260	118

istatistiksel olarak diğer algoritmaların gerisinde kalmıştır. ABC algoritması ise 92 sınıfta daha iyi olurken 75 sınıfta daha kötüdür. MGA, 94 sınıfta diğer algoritmaları geçerken, 74 sınıfta diğer algoritmaların gerisinde kalmıştır. SGA algoritmasının 79 sınıfta üstünlüğü bulunmaktadır fakat 85 sınıfta istatistiksel olarak daha kötüdür. Son olarak RS toplam 118 sınıfta istatistiksel olarak daha iyi sonuç elde ederken 260 sınıfta ise geride kalmıştır. Sonuçlar genel olarak incelendiğinde ABC algoritması SGA'yı geride bırakmıştır fakat MGA'nın gerisinde kalmıştır. MABC ise en başarılı algoritma olurken, RS ise en kötü performansı gösteren algoritma olmuştur.

Tablo 3.9 ortalama Vargha-Delaney $\hat{A}_{12}$ etki büyüklüğünü göstermektedir. Elde edilen sonuçlara göre, MGA algoritması MABC hariç diğer algoritmalarından daha iyi sonuç üretirken SGA ve ABC ise benzer performans sergilemiştir. MABC ise diğer algoritmalarından daha üstündür ve olası bir çalıştırmada diğer algoritmaları geçme olasılığı daha yüksektir.

Sonuç olarak Wilcoxon ve ortalama Vargha-Delaney $\hat{A}_{12}$ etki büyüklüğü sonucuna göre önerilen MABC algoritması istatistiksel olarak daha iyi performans sergilediği kanıtlanmıştır ve nesne yönelimli programlama için test senaryosu üretiminde başarılı bir şekilde kullanılabileceği gösterilmiştir.

3.8.6. Yakınsama ve Çeşitlilik Analizi

Bir algoritma daha hızlı yakınsamak için popülasyondaki bilgiyi kullandığında, popülasyondaki çeşitliliğin eksikliğinden dolayı erken yakınsama meydana

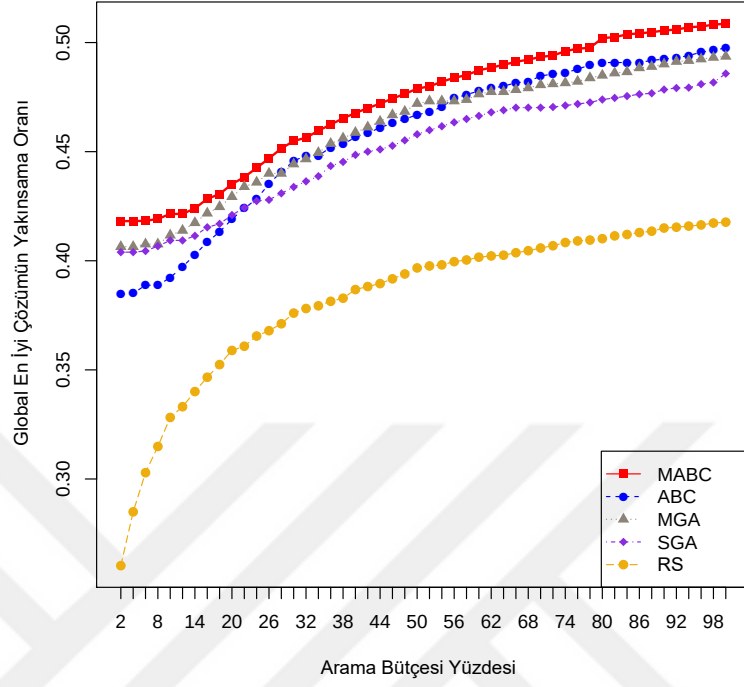
Tablo 3.9. Her bir algoritmaya ait Dal Kapsamı sonuçlarının ortalama Vargha-Delaney etki büyüklüğü $\hat{A}_{12}$ değeri.

	MABC	ABC	MGA	SGA	RS
MABC	-	0.546	0.532	0.550	0.611
ABC	0.454	-	0.481	0.498	0.569
MGA	0.468	0.519	-	0.514	0.576
SGA	0.450	0.502	0.486	-	0.563
RS	0.389	0.431	0.424	0.437	-

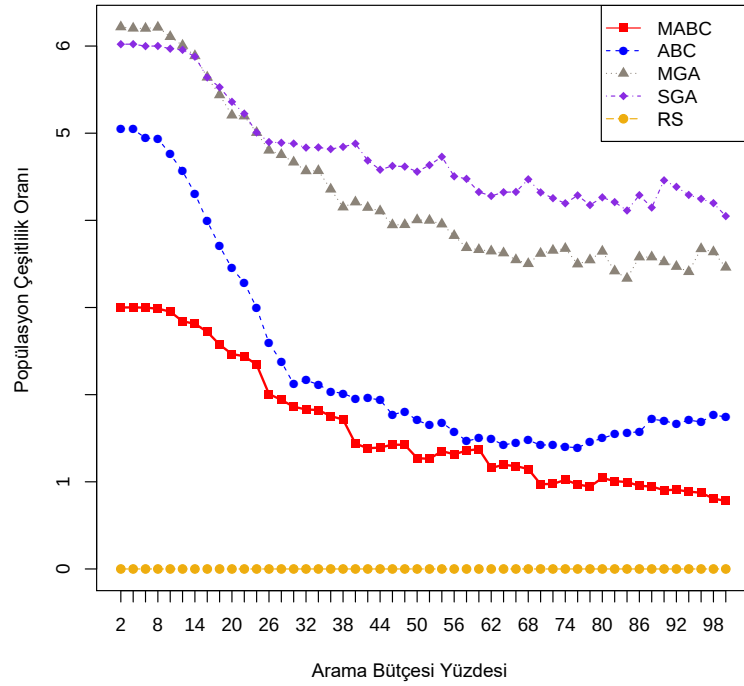
gelebilmektedir. Bu nedenle, algoritmalar verimli yakınsama operatörlerine sahip olması gerekirken, ihtiyaç duyulduğunda çeşitliliği koruyacak mekanizmalara da sahip olmalıdır. Algoritmaların keşif ve faydalanma yeteneklerini irdelemek için yakınsama ve çeşitlilik özelliklerini incelenmiştir. Ayrıntılı yakınsama ve çeşitlilik analizi Şekil 3.7 ve 3.8’de verilmiştir. Yakınsama analizi için global en iyi çözümün arama sürecinin çeşitli yüzdelğinde elde ettiği Dal Kapsamı değerleri Şekil 3.7’de verilmiştir. Verilen değerlere göre, MABC algoritmasının daha erken yakınsadığı görülmektedir. Beklendiği gibi RS en kötü yakınsamaya sahip algoritma olurken, ABC algoritması ise başlangıçta kötü bir performans sergilese de aramada sürecinde performansını iyileştirerek ikinci sıraya yükselmiştir. MGA ve SGA benzer başlangıçlar göstermiştir fakat aramanın sonunda MGA, SGA algoritmasını geçmiştir. Şekil 3.8’de ise çeşitlilik analizinin incelenmesi amacıyla, popülasyondaki bireylerin arama sürecinin çeşitli yüzdelindeki amaç fonksiyon değerinin standart sapması verilmiştir. Şekil 3.8’e göre RS’nin popülasyonu olmadığı için popülasyon çeşitliliği de sıfır olarak bulunmuştur. MGA ve SGA algoritmaları en fazla çeşitliliği koruyan algoritmalarlardır. ABC ise MABC’nin önünde üçüncü sırada bulunmaktadır. MABC algoritmasının havuz mekanizması, yakınsama hızını artırırken çeşitliliği azaltmıştır ancak yakınsama ve çeşitlilik grafikleri birlikte incelendiğinde MABC algoritmasında erken yakınsama veya yerel minimuma takılma gerçekleşmediği görülmektedir. Yani MABC algoritmasının çeşitliliğinin diğer algoritmalara kıyasla daha az olsa da yeterli olduğu görülmektedir.

3.8.7. Değerlendirme

Bu bölümde kapsama miktarını maksimum yapacak şekilde otomatik test senaryo üretimi için iki yeni çok kriterli kombinasyonel ABC algoritması önerilmiştir.



Şekil 3.7. Global en iyi çözümün yakınsaması



Şekil 3.8. Popülasyon çeşitlilik oranı

Önerilen yöntemlerde, ayırık gösterime uygun yeni yerel ve global arama operatörleri uygulanmıştır. Literatürde var olan çaprazlama tekniklerinin etkinliği incelenmiş olup, hali hazırda kapsanan hedeflerin tutulduğu arşiv mekanizması da entegre edilerek sistem kaynaklarının verimli kullanılması sağlanmıştır. Önerilen yöntemin etkinliğinin incelenmesi için literatürde en sık kullanılan algoritmalar olan MGA, SGA ve RS algoritmaları ile önerilen algoritmalar olan ABC ve MABC algoritmaları kıyaslanmıştır. İstatistiksel testlere dayanarak, MABC algoritmasının diğer algoritmalarından daha başarılı olduğu gösterilmiştir. Sonuç olarak ABC algoritması nesne yönelimli programlama için test senaryosu üretiminde etkin bir biçimde kullanılmıştır. Arşiv mekanizması performansı önemli ölçüde artırırken, önerilen değişiklikler de algoritmanın keşif ve faydalanma yeteneklerini kuvvetlendirmiştir. MABC ve ABC algoritmalarının test senaryosu üretiminde alternatif algoritmalar olabileceği gösterilmiştir. MABC algoritmasının dezavantajı ise diğer algoritmalarından daha fazla sistem kaynağına ihtiyaç duymasıdır. Bunun sebebi de yeni birey üretiminin ve havuz mekanizmasının çok fazla sistem kaynağına ihtiyaç duymasıdır. İlerleyen çalışmalarda, önerilen mekanizmaların zaman konusunda daha efektif versiyonları araştırılabilir. Ayrıca meta-sezgisel algoritmaların çeşitli kontrol parametreleri bulunmaktadır ve bu parametreler performansı etkilemektedir. Bu parametrelerin otomatik olarak ayarlanması da gelecekte planladığımız çalışmalar arasındadır. Ayrıca ceza faktörü w_i 'de algoritmaların performansı üzerinde etki sahibi olabilirler. Bunun da ayarlanması önemlidir. Çalışmaların çok fazla sistem kaynağına ihtiyaç duyması sebebiyle en geniş çalışmamızda rastgele seçilen 6569 sınıf arasından alınan sonuçlar değerlendirilmiştir. Daha fazla sınıfın değerlendirilmesi de gelecekte planlanan çalışmalar arasındadır. Ayrıca Java programlama diliyle gerçekleştirilen bu çalışmanın diğer programlama dillerinin dahil edilerek genişletilmesi planlanmaktadır.

4. BÖLÜM

RESTFUL WEB SERVİSLER İÇİN TEST SENARYOSU ÜRETİMİ

4.1. Giriş

Mikroservisler küçük, birlikte çalışan otonom servislerden oluşmaktadır. Mikroservis mimarisi ise belli bir bağlam doğrultusunda, gerçekleştirebileceği eylemlere göre düzenlenmiş, otomatikleşmiş ve geliştirilebilir mikroservislerden oluşan bir mühendislik yaklaşımıdır [180]. Mikroservis mimarisi, bir uygulamayı yüksek düzeyde bakım yapılabilir servisler topluluğuna dönüştürür ve monolitik mimarinin karmaşıklığını azaltır. Ancak mikroservis mimarisinin Otomatik Hizmet Keşfi , Aşırı Geç Bağlanma gibi özellikleri entegrasyon testini engellerken, kullanım başına maliyet birim testini ve Hizmet Kalitesi testini etkilemektedir [9, 10]. Bu nedenle mikroservis mimarisi uygulanarak gerçekleştirilmiş uygulamaların testi oldukça zor ve yüksek maliyetlidir. Bu yüksek maliyeti düşürmenin yollarından biri de yazılım testlerinin otomatik olarak üretilmesidir.

Pek çok mikroservis temsili durum aktarımı (Representational State Transfer, REST) [181] veya basit nesne erişim protokolü (Simple Access Protocol, SOAP) gibi yaklaşımlar ile tasarlanmaktadır. Hiper-Metin Transfer Protokolü (Hypertext Transfer Protocol, HTTP) kullanarak çeşitli veri formatını desteklemesi sebebiyle genelde REST, SOAP'tan daha fazla tercih edilmektedir.

REST yapısı [181] ilk olarak 2000 yılında Roy Fielding tarafından tanımlanmıştır. REST, HTTP gibi tercih edilen bir protokole dayalı bir dizi mimari kılavuz görevi görmektedir. Bir sunucu/istemci uygulamasının "RESTful" olarak kabul edilebilmesi için durumsuz olması ve belirtilen sınırlamaları karşılıyor olması gerekmektedir. Aynı

zamanda gösterim bir Tekdüzen Kaynak Tanımlayıcı (Uniform Resource Identifier, URI) tarafından tanımlanmış olmalıdır ve gösterimin kaynağının istemciye gerçek veriden bağımsız olarak JavaScript Nesnesi Gösterimi (JavaScript Object Notation, JSON) veya Genişletilebilir İşaretleme Dili (Extensible Markup Language, XML) yoluyla iletilmesi gerekmektedir. Kaynaklar da uygun HTTP fiiliyle (GET, POST, DELETE, vs.) kontrol edilmelidir [143].

RESTful web servislerin otomatik olarak test edilmesi yazılım bakım maliyetlerini düşürmekte ve manuel test etme sonucunda ortaya çıkan insan faktörüne bağlı hata yapma ihtimalini azaltmaktadır. Bu bölümde mikroservis mimarisinin yapı taşları olan RESTful web servislerin otomatik test senaryosu üretim metodolojisi anlatılmış ve önerilen algoritmaların ayrıntılarından bahsedilmiştir.

RESTful web servisler genelde HTTP protokolüne dayalı çalışmaktadır. HTTP, bir dizi kural ve kısıtlamaya dayalı olarak Dünya Çapında Ağ (World Wide Web, WWW) üzerindeki etkinlikleri kontrol eden bir uygulama protokolüdür. Bir HTTP mesajı, kaynak yolu, metot/fiil, başlık ve gövdeden [143] oluşmaktadır. Bir HTTP talebi gönderildiği zaman HTTP cevabı sunucu tarafından geri gönderilmektedir. Bu HTTP cevabı da bir başlık, gövde ve üç haneli durum kodundan meydana gelmektedir. Bu durum kodları 1xx (Bilgilendirici), 2xx (Başarılı), 3xx (Yönlendirme), 4xx (İstemci Hatası) ve 5xx (Sunucu Hatası) olmak üzere beş farklı türde olabilmektedir. HTTP protokolü durumsuzdur (stateless) ve geçmiş bilgileri ihtiva etmez. Yani bütün gerekli bilgilere, HTTP isteğinin içerisinde ihtiyaç duyulmaktadır.

4.2. Önerilen Algoritmaların Bileşenleri

RESTful web servislerin test edilmesi için de test senaryosu adı verilen bir dizi HTTP isteğinin, doğru sırada, doğru parametreler ile çalıştırılması gerekmektedir. Sonraki alt bölümlerde RESTful test senaryosunun temsil biçimi, üretilen testlerin kalitesini ölçmek için kullanılan amaç fonksiyonu, doğrulayıcı tasarımı ile algoritmaların yeni çözüm üretim mekanizması ve arama operatörleri anlatılmıştır.

4.2.1. Test Senaryosunun Temsil Biçimi

RESTful web servislerin testinde son kullanıcıya sunulan son çözüm bir veya daha fazla test senaryosundan oluşan bir test paketidir (test suite) ve test senaryosu üretilirken temel amaç, her bir hedef için ayrıca tutulan test senaryosunun iyileştirilmesidir. Her bir test senaryosu bir veya daha fazla HTTP talebinden meydana gelmektedir. Test senaryosu üretilirken, bu HTTP talebinin her bir parametresi rastgele olarak oluşturulmaktadır. Bu parametreler HTTP fiili, başlık, yol parametreleri, sorgu parametreleri ve gövde yükünden oluşmaktadır [143].

Bir HTTP talebi oluşturmadan önce ilgili Uygulama Programlama Arayüzü (Application Programming Interface, API) hakkında bilgi sahibi olunması gerekmektedir. RESTful web servisler için oluşturulmuş olan Swagger¹ isimli araç bu bilgiyi sağlayabilmektedir. RAML² gibi daha az popüler farklı araçlar da bulunmaktadır fakat literatürde Swagger aracı daha yoğun kullanılmaktadır. Swagger aracı API hakkındaki bilgiyi JSON formatında kullanıcıya sunmaktadır.

Örnek bir Swagger tanımı Şekil 4.1'de gösterilmektedir. Bu Swagger tanımını indirip ayrıştırdıktan sonra, her bir API uç noktası için taslak kromozomlar oluşturulmaktadır. Bu taslak kromozomlar, API'nın IP adresi, gerekli HTTP başlıkları gibi sabit genler ve bir dizi değiştirilebilir genlerden oluşmaktadır. Bu değiştirilebilir genler basit genlerin birleşiminden oluşan karmaşık veri tiplerinden meydana gelmektedir. Bu veri tipleri aşağıda verilmiştir [143].

- Array: Aynı genleri barındıran dizileri temsil etmektedir. Çok büyük test senaryolarından kaçınmak için üst sınırı tanımlanmalıdır.
- Base64String: Base-64 formatında kodlanmış string ifadeleri temsil etmektedir.
- Boolean: True ve False değerlerini temsil eden alandır.
- CycleObject: Çizgelerin JSON gösterimi ile ilişkili sonsuz döngü problemini ortadan

¹<https://swagger.io>

²<https://raml.org>

kaldıran alandır. Bir nesne türüyle ikinci kez karşılaşıldığında kullanılır.

- Date: Gün, ay ve yıldan olmak üzere üç farklı Integer genden oluşan veri tipidir. Geçerli ve geçersiz değerler ile sınırlandırılmıştır. Örneğin ay değeri 1 ve 12 aralığındadır.
- Time: Bu gen saat, dakika ve saniyeyi ifade eden 3 Integer gen içerir. Date tipinde olduğu gibi geçerli ve geçersiz değerlerle sınırlandırılmıştır.
- DateTime: Date ve Time olmak üzere iki farklı genden oluşmaktadır.
- Disruptive: Bir genin mutasyona uğrama olasılığını belirten gendir.
- Double: Double tipteki sayıları temsil eden gendir.
- Enum: Sabit olasılıklar kümesinden bir değeri temsil eden (tipik olarak dizeler veya tamsayılar) veri tipidir.
- Float: Kayan noktalı sayıları temsil etmektedir.
- Integer: Tam sayıları temsil etmektedir.
- Long: Long tipteki sayıları temsil etmektedir.
- Map: String gibi aynı tipteki genlerin haritalandırma tipini temsil etmektedir.
- Object: JSON nesnesini temsil etmektedir.
- Optional: Fenotipte varlığı bir boole değer tarafından kontrol edilen başka bir geni içeren bir geni temsil etmektedir. Örneğin isteğe bağlı sorgu parametrelerini temsil etmek için gereklidir.
- String: Minimum ve maksimum uzunluk dahilinde sınırlanmış bir karakter dizisini temsil eder.

```

1  "/products/{productName}/configurations/{configurationName}/features/{featureName}" :
2  {
3    "post" : {
4      "operationId" : "addFeatureToConfiguration",
5      "produces" : [ "application/json" ],
6      "parameters" : [
7        {
8          "name" : "productName",
9          "in" : "path",
10         "required" : true,
11         "type" : "string"
12       },
13       {
14         "name" : "configurationName",
15         "in" : "path",
16         "required" : true,
17         "type" : "string"
18       },
19       {
20         "name" : "featureName",
21         "in" : "path",
22         "required" : true,
23         "type" : "string"
24       },
25       {
26         "name" : "productName",
27         "in" : "path",
28         "required" : true,
29         "type" : "string"
30       },
31       {
32         "name" : "configurationName",
33         "in" : "path",
34         "required" : true,
35         "type" : "string"
36       }
37     ],
38     "responses" : {
39       "default" : {
40         "description" : "successful operation"
41       }
42     }
43   }
44 }

```

Şekil 4.1. Bir kaynağa ait POST sorgusunun örnek Swagger tanımı.

4.2.2. Amaç Fonksiyonu

Rastgele oluşturulan test senaryolarının kalitesini belirleyerek arama algoritmasına rehberlik edecek bir amaç fonksiyonuna ihtiyaç bulunmaktadır. Bu amaç fonksiyonu tasarlanırken de test senaryosunun kapsadığı hedefler dikkate alınmaktadır. Her bir test senaryosu bir veya birden fazla hedefi kapsayabilmektedir. Bu hedefler için üretilen test senaryolarının tutulduğu ve arşiv adı verilen harici bir popülasyonlar topluluğu bulunmaktadır. Her bir t hedefi için ayrı tutulan popülasyonunun bireylerinin kalitesi yalnızca t hedefi için toplanan sezgisel değerlere göre belirlenmektedir. Her bir hedefe

ait $[0, 1]$ aralığında sezgisel amaç fonksiyonu değeri bulunmaktadır. Kapsanan hedeflerin sezgisel amaç fonksiyon değeri 1 olurken, 1'den küçük değerler ise ilgili hedefin henüz kapsanmadığını belirtmektedir.

Test senaryolarının, test edilen sistemin komutlarının kapsamı, bytecode seviyesinde dalların kapsamı ve HTTP durum kodları olmak üzere üç farklı kapsanması gereken hedefi bulunmaktadır.

4.2.2.1. Komut Kapsamının Elde Edilmesi

Yüksek kapsamlı test senaryoları üretebilmek için ilgili test senaryosunun elde ettiği kapsama miktarının ölçülebilmesi gerekmektedir. Aksi taktirde hangi test senaryosunun daha yüksek kapsama sahip olduğu bilinemez. HTTP durum kodu kolayca elde edilebilir olsa da satır ve dal kapsamı gibi saydam kutu metriklerine erişebilmek için kaynak koda erişim gerekmektedir. Dolayısıyla bir SUT başlatıldığında prob eklenmekte ve bu kapsama metrikleri elde edebilmektedir.

4.2.2.2. Bytecode Seviyesinde Dal Kapsamının Elde Edilmesi

Bir testin ne kadar kapsama ulaşabildiğini elde etmek tek başına yeterli değildir. Bir test senaryosunun kodun ne kadar yüzde ile kapsadığını bilmek, nasıl daha fazla kapsama miktarına erişilebileceği hakkında bilgi vermemektedir. Genellikle bir kodun kapsanamama sebebi if blokları içerisinde bulunmasıdır. Rastgele girdilerin bu karmaşık if bloklarını çözebilmesinin olasılığı oldukça düşüktür. Bu problemin en etkin çözüm yöntemlerinden biri, test verisinin çözüme ne kadar yaklaştığını gösteren sezgisel değerin atanmasıdır. Örneğin Şekil 4.2'de verilen kod parçacığı ele alındığında, 3 farklı dal (if, else if ve else) bulunmaktadır. Bu dallardan if bloğuna ait $x == 500$ kısıtı düşünüldüğünde 600 sayısı 5 sayısından daha yakın bir çözümdür. Bu durumu belirlemek için literatürdeki en bilinen yöntem ise dal mesafesidir [51, 57]. Dal mesafesinin fonksiyonları Tablo 4.1'de verilmiştir.

Dal mesafesinin ölçülmesi, dal kapsamının yükseltilmesi konusunda literatürdeki en iyi yaklaşımlardan biridir [143]. Ancak dal kapsama metriği, kodda herhangi bir istisna meydana gelmesi durumunda satır kapsamı için pek de iyi bir seçenek olmamaktadır.

Tablo 4.1. Dal mesafesi fonksiyonu.

Karşılaştırma Betimi	Amaç Fonksiyonu
$a = b$	$\text{if } abs(a - b) = 0 \text{ then } 0 \text{ else } abs(a - b)$
$a \neq b$	$\text{if } abs(a - b) \neq 0 \text{ then } 0 \text{ else } abs(a - b)$
$a < b$	$\text{if } a < b \text{ then } 0 \text{ else } a - b$
$a \leq b$	$\text{if } a \leq b \text{ then } 0 \text{ else } a - b$
$a > b$	$\text{if } a > b \text{ then } 0 \text{ else } b - a$
$a \geq b$	$\text{if } a \geq b \text{ then } 0 \text{ else } b - a$

```

1
2   if (x==500) {
3       ...
4   } else if (x > 1000) {
5       ...
6   } else {
7       ...
8   }

```

Şekil 4.2. Örnek bir if-else bloğu.

Örneğin 10 satırlık bir program ele alındığında A test senaryosu 3. satırda istisna meydana getirirken, B test senaryosu 8. satırda meydana getirebilmektedir. Fakat dal mesafesi metriği B test senaryosunun A'dan daha iyi olduğunu söyleyememektedir. Bu konunun da amaç fonksiyonunda ele alınabilmesi için bir sezgisel değer bulunmaktadır ve ilk başta $h = 0.5$ olarak atanmaktadır. Eğer bir test senaryosu istisna ortaya çıkarmazsa $h = 1$ olmaktadır. Eğer ilgili komuta hiç erişilemediyse $h = 0$ olmaktadır. Örneğin, her iki test senaryosu da (A ve B) çalıştırıldığında; A test senaryosu ilk 2 satırda 1 ve 3. satırda 0.5 olurken, B test senaryosu, 1. ve 7. satırlar arası 1 olurken 8. satır 0.5, 9 ve 10. satırlar ise 0 değerini alacaktır. Bu da B test senaryosunun A test senaryosundan daha iyi olduğunun söylenebilmesini sağlamaktadır.

4.2.2.3. HTTP Durum Kodlarının Elde Edilmesi

Döndürülen HTTP durum kodlarının amaç fonksiyon değerinde ara değerler bulunmamaktadır. Bu durum kodları ya kapsanır ($h = 1$) veya kapsanmaz ($h = 0$) olarak atanmaktadır. Bu durum kodları yeni test hedefi olarak modellenmektedir ve arama sürecinde elde edilen diğer durum kodlarını kapsayan testler ilgili hedef popülasyonuna kaydedilmekte ve böylece bu testlerin kaybedilmesi önlenmektedir. Eğer 5xx durum kodu cevap olarak döndüyse bu sunucu tarafında veya harici servislerde (veritabanı gibi) bir problem yakalandığını belirtmektedir. Bu hata kodunu hedef olarak belirlemek hata

tespitinin en kolay yöntemlerinden biridir. Bütün hatalar bu kod ile yakalanamamaktadır fakat başlangıç için gayet uygundur.

4.2.2.4. Test Hedeflerinin Bilgilerinin Sunulması

Literatürdeki saydam kutu çalışmalarının büyük bir çoğunluğu birim test üretmeyi hedeflemektedir. Ancak entegrasyon/sistem testi üretiminin gerçek dünya problemleri için ölçeklenebilir olmasını sağlamak için bazı zorlukların üstesinden gelmek gerekmektedir. Örneğin bytecode manipülasyonu kullanarak kod kapsamı ve dal mesafesi ölçülebilse de bu değerin nasıl alınacağı konusunda çeşitli problemler bulunmaktadır. Birim testinde bir test senaryosu üretim aracı, SUT içine entegre edilen aynı işlemde çalışabilmektedir böylece değerler doğrudan elde edilebilir olacaktır. Test eylemi yalnızca aracın yazılmış olduğu dil ile sınırlandırılmış olacağı için bu yaklaşım optimal değildir. Ayrıca test aracı ile SUT içindeki üçüncü parti uygulamalar arasında çeşitli çakışmalar da olabilmektedir. Ayrıca RESTful web servis testinde senaryolar dilden bağımsız HTTP çağrılarında oluşacağı için tek bir dile odaklanmak da gereksiz bir sınırla olacağı için test aracı ve SUT'un farklı işlemlerde çalışmasını sağlamak daha uygun olacaktır. SUT çalıştırıldığında, kodu otomatik olarak gösteren bir kütüphane sağlanırsa, kütüphanenin kendisi otomatik olarak bir RESTful API sağlayacak ve tüm kapsama miktarlarını ve dal mesafesi bilgilerini JSON formatında sunacaktır. Test aracı, testler oluşturulurken ve çalıştırılırken, testlerin uygunluğunu belirlemek için bu API'yı kullanmaktadır. Böylelikle aynı test aracı birden fazla programlama dilinde tasarlanmış API'ları test etmek için kullanılabilir. Yalnızca bu SUT'ların, bu bilgiyi sağlayacak kod yönlendiricilerine ihtiyacı bulunmaktadır. Bu yaklaşım birim testinde performansı yüksek bir yaklaşım değildir. Farklı bir işleme yapılan HTTP çağrısının maliyeti, birim testi gerçekleştirmenin maliyetine oranla oldukça büyüktür fakat sistem seviyesinde test gerçekleştirilirken her bir test çalıştırmasında bütün uygulama çalışır. Bu nedenle özellikle karmaşık ve harici servislerle (veritabanı gibi) etkileşime giren sistemlerde bu tür bir yaklaşım daha yönetilebilir olacaktır [143]. Sisteme gelen bu ekstra yükün daha yönetilebilir olması için aşağıdaki iki eylem ile kontrol altında alınmaktadır.

- SUT başlatıldığında, geliştiricinin hangi paketleri test edeceği belirtilmelidir. Bütün sınıfların bilgisinin sunulması oldukça verimsiz olacaktır. Örneğin tomcat gibi

sunucular veya hibernate gibi orm kütüphanelerinin kapsam metriklerini elde etmek gereksizdir.

- Varsayılan olarak, SUT'a ait kod kapsamı veya dal mesafesi bilgileri sorgulanırken tüm bilgiler alınmamaktadır. Sadece yeni kapsanan hedefler veya daha iyi dal mesafesine sahip olanlar geri döndürülür. Bunun sebebi eğer SUT binlerce satır koddan oluşuyor ise binlerce satır içeren JSON verisinin çözümlenmesi zaman alacağından test aracı hangi hedef için bilgiye ihtiyaç duyduğunu sormaktadır.

Örnek olarak aşağıdaki gibi bir HTTP isteğinin gerçekleştirilmesi incelendiğinde:

```
POST /products/s07atT07/constraints/excludes
```

HTTP isteğinin çalışması ile birlikte test senaryosunun elde ettiği sonuç Java Agent ve bytecode manipülasyonu ile toplanmaktadır ve kod yönlendirici tarafından RESTful API aracılığı ile, Şekil 4.3'te olduğu gibi JSON formatında, hangi hedeflerin ne kadar kapsandığını belirten bir rapor sunulmaktadır.

Buradaki "*descriptiveId*" alanı hangi hedef için elde edilen sonuç olduğunu belirtmektedir. "*value*" alanı ise bu hedef için hesaplanan amaç fonksiyon değerini vermektedir. Eğer bu değer 1 ise o hedefin kapsandığını göstermektedir.

4.2.3. Doğrulayıcı (Oracle)

Saydam kutu yaklaşımında test senaryoları oluşturulurken doğrulayıcının ne olması gerektiği ile ilgili bir problem bulunmaktadır. Bir test senaryosunun sonucunun doğru olup olmadığını söyleyen bir işlev, doğrulayıcıdan biri olarak düşünülebilmektedir. Manuel gerçekleştirilen testlerde, geliştiriciler belli bir test senaryosu için beklenen sonucun ne olması gerektiğine karar vermekte ve doğrudan bu beklenen sonuca göre testleri yazmaktadır. Yüzlerce test senaryosunun otomatik olarak üretildiği durumda ise geliştiricilerden bu beklentiye yazmalarını istemek uygun bir seçenek değildir [143] ve bu problemin basit bir çözümü de bulunmamaktadır [182]. Sistem seviyesinde testlerde ise en belirgin doğrulayıcı, test yapılırken sistemin çöküp çökmediğinin kontrol edilmesidir. Bir test senaryosu yazılımda hata tespit edebilir fakat tüm hatalar uygulamanın çökmesine sebep olmamakta; küçük bir bölümü uygulamanın tamamen çökmesine sebep olmaktadır.

Birim testinde, test edilen sınıflarda veya yöntemlerde fırlatılan istisnalara bakılabilmektedir [183]. Ancak buradaki önemli problemlerden biri de fırlatılan istisnanın bir hata belirtisi değil daha çok bir ön koşulun (null girdisi sebebiyle ortaya çıkan NullPointerException gibi) ihlal olması sebebiyledir [143].

```

1  "data": {
2    "targets": [
3      {
4        "id": 0,
5        "descriptiveId": "Line_at_org.javiermf.features.CORSFilter_00022",
6        "value": 1,
7        "actionIndex": 0
8      },
9      {
10       "id": 1,
11       "descriptiveId": "Class_org.javiermf.features.CORSFilter",
12       "value": 1,
13       "actionIndex": 0
14     },
15     {
16       "id": 2,
17       "descriptiveId": "Line_at_org.javiermf.features.CORSFilter_00023",
18       "value": 1,
19       "actionIndex": 0
20     },
21     {
22       "id": 3,
23       "descriptiveId": "Line_at_org.javiermf.features.CORSFilter_00024",
24       "value": 0.5,
25       "actionIndex": 0
26     },
27     {
28       "id": 4,
29       "descriptiveId": "Success_Call_at_org.javiermf.features.CORSFilter_00024_0",
30       "value": 1,
31       "actionIndex": 0
32     }
33   ]

```

Şekil 4.3. Test senaryosu çalıştırıldıktan sonra üretilen örnek JSON formatında rapor.

Otomatikleşmiş bir doğrulayıcı bulunmasa bile üretilen testler regresyon testi için yine de faydalı olmaktadır. Örneğin Şekil 4.4'teki gibi bir test senaryosunu ele aldığımızda test fonksiyonuna 25 verdiğimizde beklenen sonucun 97 olduğu görülmektedir. Bunun gibi otomatik olarak üretilmiş bir test senaryosunun mevcut test senaryoları arasına alınması ile test edilen fonksiyonunun davranışının değişiklik göstermesi durumunda, sürekli entegrasyon (continuous integration) aracının çalışması ile bu test başarısız olacaktır. Bu da son zamanlarda yapılan değişikliklerin gerçekten bu fonksiyonun işlevini bozup bozmadığını veya bu işlevin geçerli olup olmadığını anlamak için buradaki davranış değişikliğinin sebebinin aranmasını sağlayacaktır [143].

```

1  int x = 25;
2  int res = test(x);
3  assertEquals(97, res);

```

Şekil 4.4. Birim test örneği.

Bu birim testler gibi RESTful web servisler için üretilen test senaryoları da regresyon testi için kullanılabilir. Özellikle güvenlik açısından oldukça faydalıdır. Örneğin 403 numaralı yetkisiz durum kodu olduğu bir HTTP çağrısının yetkilendirme kontrolünün yanlış gevşetildiği durumlarda tespit edilmesini sağlayabilmektedir. Ayrıca durum kodları otomatik doğrulayıcı olarak kullanılabilir. 4xx kodu üreten bir istek hata anlamına gelmezse de, 5xx'li durum kodları genellikle hizmetlerin birinde meydana gelen hata sebebiyle ortaya çıkmaktadır. Bunun en tipik örneklerinden biri de fırlatılan istisnalar. Eğer API bir istisna yakalarsa sunucu çökmekte ve bu istisnayı 5xx'li koda dönüştürmektedir. Ancak SUT'un doğru bir şekilde tasarlanması gerekmektedir. Örneğin kullanıcı geçersiz bir giriş gönderirse, 5xx kodunu değil 4xx kodunu alması gerekir. Eğer girdilerin doğrulanması yapılmaz ise veya uygulamanın istisna fırlatmasına izin verilmezse, iki temel problem ortaya çıkmaktadır. Bu problemlerden ilki, isteği yapan kullanıcı kendi hatası olduğunu bilemeyecek ve geliştirici de 5xx hatasını nasıl düzelteceğini bilemeyecektir. İkincisi ise uygulama harici kaynaklar (veritabanı gibi) üzerinde bir dizi işlem gerçekleştirdiğinde, bütün eylemler tamamlanmadan bir istisna meydana gelirse uygulama tutarsız bir konumda kalır ve tüm sistemin yanlış çalışmasına sebep olabilir. Örneğin sistemde veritabanında kayıtlı olmayan bir id'ye ait kayıtları getirmesi istendiğinde bu id'nin var olup olmadığı kontrol edilmez ise sistem, 404 hatası yerine NullPointerException fırlatacaktır ve bu da 500 server hatasına sebep olacaktır. Bu nedenle geçersiz kimliğe sahip bir test oluşturulduğunda null pointer hatası ile sonuçlanacaktır. Böyle bir istisna HTTP yanıtı oluşturacak olan sunucuya yayılacaktır [143].

Amaç fonksiyonu her SUT uç noktası için yeni HTTP durum kodlarının kapsanmasını ödüllendirdiği için 5xx durum kodlu testleri de arşivde saklar. Ancak bu tür testlerin nasıl fayda sağlayacağı ve kullanıcıya nasıl sunulacağı henüz yeteri kadar araştırılmış bir konu değildir. Bir sunucunun asla 5xx hata vermemesi gerekmektedir. Ancak 5xx her zaman bir sunucu hata belirtisi olmadığından bu durumun yakalanması ve kayıt altına alınması

tercih edilmektedir. Bu nedenle 5xx hatası ile karşılaşıldığında, sistemin gerçekten de 5xx hatası vermesi gerektiği kabul edilir ve ona göre test senaryosu kaydedilir. Bu türde bir test senaryosu kullanıcıya çıktı olarak verildiğinde sistemde hataya sebep olabilir ve bu da diğer test senaryolarından daha değerli hale gelmesi ile sonuçlanabilir. Bu nedenle 5xx hata kodlarını kontrol etmek, RESTful web servisler için kısmi bir doğrulayıcı tanımlamanın en kolay yoludur. Her türlü hatayı yakalayamayacak olsa da makul bir başlangıçtır [143].

4.2.4. Yeni Çözüm Üretimi

Arama algoritmalarının yerel minimuma takılmaması için rastgele örnekleme yapabilir olması gerekmektedir. Bir test senaryosu rastgele örneklendiğinde içerisinde 1 ile n aralığında değişen HTTP çağrısı bulunmaktadır. Her HTTP çağrısının parametreleri de rastgele oluşturulmaktadır. Bir API için hangi uç noktaların bulunduğu Swagger gibi araçlarla elde edilmektedir. Yeni testlerin örneklenmesi, bu araçlardan gelen kısıtlamalara göre gerçekleştirilmektedir. Bu rastgele dizelerin geçerli bir HTTP çağrısı oluşturması oldukça düşük bir olasılık olduğundan HTTP çağrılarının örneklenmesi için tamamen rastgele dizelerin kullanılması uygun değildir.

Bir test senaryosu örneklendikten sonra, arama süreci içerisinde yeni HTTP çağrısı eklemesi, kaldırılması veya mevcut çağrılardan birinin parametrelerinin değiştirilmesi gibi çeşitli mutasyonlara uğramaktadır. Bu testlerin gelişimi nesne yönelimli programlamada olan test senaryosu üretimine benzese de arada büyük bir fark vardır. Örneğin Şekil 4.5'teki kod parçacığı ele alındığında, targetMethod isimli sınıfı test etmek için sadece x girişini değil sınıfın dahili durumunun da değiştirilmesi gerekebilmektedir. Bu dahili durum bar değişkeninde bulunmaktadır. Bir test aracı bu bar değişkenine yeni çağrıda bulunabilir veya tamamen kaldırabilir durumdadır. Foo ve Bar sınıflarındaki yöntemlerin içeriği bilinmese bile bar'ı çağırarak için kullanılan yöntemler targetMethod'u test etmek için faydalı olabilmektedir. Ancak RESTful web servis testinde durum tamamen farklıdır. Şekil 4.6'da bir çağrı dizisi örneği verilmiştir. Bu uç noktayı tam olarak test edebilmek için 100 numaralı id'ye sahip bir nesneye ihtiyaç duyulduğu için buradaki GET çağrısı muhtemelen 404 Not Found mesajı ile karşılaşacaktır. Fakat bu kaynağı oluşturmak için herhangi bir işlemin bulunma

olasılığı düşüktür. Rastgele örneklenmiş bir test büyük ihtimalle "POST /products" çağrısında bulunurken 100 id'li bir nesne oluşturmayacaktır. Hatta sistem otomatik artımlı indeksleme kullanıyor olabileceği için doğrudan belli bir id'ye bağlı nesne oluşturması mümkün olmayabilir. Eğer ki şans eseri bu ürün oluşturulmuş olsa bile aynı testin iki sefer çalışmasında farklı sonuçlar (ürünün değiştirilmesi gibi) elde edilebilmektedir. Bu nedenle dal mesafesi tabanlı standart amaç fonksiyonları, çeşitli veritabanı işlemlerinde aramaya rehberlik etme konusunda yeterli olmayabilmektedir.

```

1  Foo foo = new Foo();
2  foo.something ();
3  foo.somethingElse ();
4  Bar bar = new Bar();
5  int x = 42;
6  bar.targetMethod(x);

```

Şekil 4.5. Nesne yönelimli test senaryosu örneği

```

POST /products
POST /products
GET /products/100

```

Şekil 4.6. HTTP çağrı örneği

Bu nedenle yeni test senaryosu rastgele örneklenirken belirli bir P olasılığı ile bu testin yapısının önceden tanımlandığı bir şablon kullanılabilmektedir [143]. SUT içerisinde rastgele bir uç nokta ve mevcut HTTP yöntemlerinden biri seçilir. Bu çağrıyı gerçekleştirmek için ise SUT'un uygun konuma getirilmesi için çeşitli şablonlar eklenmektedir. Ne tür HTTP çağrılarının ekleneceği ise HTTP çağrısının fiiline ve URI'sine bağlıdır. Bu şablonlar aşağıda verilmiştir.

GET Şablonu: Son HTTP isteğinin yol parametresinin değişken mi yoksa sabit mi olacağı olmak üzere iki temel durum bulunmaktadır.

```
GET /products
```

Yukarıdaki komut ele alındığında, bu uç nokta bir veya birden fazla kaynağı temsil edebilmektedir. Bir GET isteğini düzgün test edebilmek için birkaç adet girdiye ihtiyaç duyulmaktadır. Bu nedenle rastgele k adet POST çağrısı eklenir ve bu şablonda, aşağıda gösterildiği gibi k sayıda (örnekte 3 adet) POST ve en son da bir adet GET bulunur.

```
POST /products
```

```
POST /products
POST /products
GET /products
```

Eğer uç nokta POST/PUT taleplerine izin vermiyor ise $k = 0$ olmaktadır.

Bir diğer durum da aşağıdaki istek gibidir.

```
GET /products/{id}
```

Bu durumda Swagger şemasında herhangi bir oluşturma işlemi olup olmadığı (POST veya PUT) kontrol edilir. Eğer yoksa, */products* içeren URI'deki en yakın ata kaynaklarda bir oluşturma işlemi olup olmadığı kontrol edilir. Eğer oluşturma işlemi varsa aşağıda görüldüğü gibi önce ekleme işlemi ardından da bu eklenen veriyi çekme işlemi gerçekleştirilmektedir.

```
POST /products
GET /products/{id}
```

POST ile oluşturulan kaynağın GET ile nasıl ilişkilendirildiği ise Zincirleme Konumlar kısmında açıklanmaktadır.

POST Şablonu: POST, kaynak oluşturmak için kullanıldığı için özel olarak ele almaya gerek yoktur. Yalnızca tek bir POST çağrısı yeterlidir.

PUT Şablonu: PUT bir kaynağı tamamen değiştiren idempotent (tek sefer etki gösteren) bir fiildir. Böyle bir kaynak yoksa PUT metodu ile oluşturulabilir veya 4xx hatası dönebilir. Tek bir PUT ile bir kaynak oluşturulması veya POST ile önce bir kaynak oluşturma ardından PUT ile bu kaynak üzerinde değişiklik yapmak olmak üzere iki farklı durum ele alınmaktadır.

PATCH Şablonu: PATCH bir kaynak üzerinde kısmi güncelleme yapmaktadır. Bu nedenle test etmek için önce bu kaynağı yaratmak gerekmektedir. PATCH, PUT yönteminden farklı çalışmaktadır. Eğer kaynak eksikse PUT bu kaynağı oluşturabilmekteyken PATCH oluşturamamaktadır. Bu nedenle önce POST/PUT ve ardından PATCH çağrılması gerekmektedir. Kısmi güncellemelerin doğru çalışıp çalışmadığını kontrol etmek için aynı kaynak üzerinde birden fazla PATCH yapılması gerekebildiğinden bir şablonda bir veya iki PATCH olacağı rastgele olarak belirlenmektedir.

```
POST /products
```

```
PATCH /products/{id}
PATCH /products/{id}
```

DELETE Şablonu: Bir DELETE filinin doğru test edilebilmesi için kaynağın var olması gerekmektedir. Bu nedenle önce POST/PUT ile kaynak oluşturulur ve DELETE ile silinmektedir.

Ara Kaynak: Önceki şablonlarda bir eylemi test edebilmek için önce kaynak yaratmak gerekmektedir. Fakat bir kaynak bir başka kaynağa bağlı olabilmektedir (*GET /products/id/price* gibi). Böyle bir GET çağrısı muhtemelen 404 hatasına sebep olacaktır. Bunu engellemek için ara kaynakların tamamının tek tek oluşturulması gerekmektedir. Bu nedenle *GET /products/id/price* gibi bir isteğin çalışabilmesi için önce *GET /products/id* isteğinin oluşturulması gerekir. Bu durumda istek şöyle bir test olacaktır:

```
POST /products
GET /products/{id}/price
```

Daha karmaşık bir örnek olan *GET /products/pid/subitems/sid* test edilmek istendiğinde ise test şöyle olacaktır:

```
POST /products/{pid}/subitems
GET /products/{pid}/subitems/{sid}
```

Ancak POST da henüz mevcut olmayan bir alt kaynağa bağlı olduğu için o alt kaynak da üretilmelidir ve testin son hali şöyle olacaktır:

```
POST /products
POST /products/{pid}/subitems
GET /products/{pid}/subitems/{sid}
```

Zincirleme Konumlar: Bir üstteki örnek ele alındığında, buradaki ikinci POST'ta kullanılan pid'nin ilk POST ile oluşturulan kaynak ile aynı olması ve GET'teki pid ile sid'nin bu id'lerle aynı olması sağlanmalıdır. API bu tür kaynakları doğrudan oluşturmaya izin veriyor ise aşağıdaki gibi bir test oluşturulabilmektedir.

```
POST /products
POST /products/{pid}/subitems
GET /products/{pid}/subitems/{sid}
```

Buradaki id'ler el ile belirlenebilmektedir ve bu id'ler mutasyona uğratılamaz. Ancak POST ile bir kaynak oluşturulduğunda önerilen yaklaşım (RFC 7231), bu tür bilgilerin HTTP yanıtının Location Header bilgisinde gönderilmesidir. *POST /products/100* oluşturulduğunda *location: /products/100* belirtilir. Bu sayede pid ve sid değişkenlerini

sabit kodlamak yerine bunları bu tür kaynaklardan dinamik olarak elde edilebilmektedir. Dolayısıyla, ikinci POST'ta bulunan yol parametresi birinci POST'un cevabına bağlı olacaktır. GET'in yol parametresi ise ikinci POST'un cevabına bağlı olacaktır. Ayrıca bazı uç noktalarda bu bilgiler HTTP yanıtının yükü olarak gönderilmektedir. Eğer bu yük JSON veya XML gibi formatlar ile yapılandırıldıysa, uç noktadaki yol parametrelerinden biri ile ilişkili (id gibi) olup olmadığı kontrol edilmekte ve sezgisel olarak POST cevabından çıkarılmaya çalışılmaktadır. Tabii bu yöntem, ilgili değerleri bulmayı garanti etmez fakat location header yok ise bu yöntemi denemek de faydalı olabilmektedir.

Sonuç olarak bu şablonlar ile test yapısının önceden tanımlandığı akıllı bir örnekleme sağlanabilmektedir. Şekil 4.7'de rastgele örnekleme kullanılarak, Şekil 4.8'de ise akıllı örnekleme kullanılarak oluşturulan test senaryo örnekleri verilmiştir.

```

1  GET /products/GEWJ
2  POST /products/ILiv402z/features/Z
3  GET /products
4  DELETE /products/ED/constraints/45721
5  GET /products/M/features

```

Şekil 4.7. Rastgele örnekleme ile oluşturulan test senaryosu.

```

1  POST /products
2  POST /products/WLiv402z/features
3  GET /products/WLiv402z/features/U

```

Şekil 4.8. Akıllı örnekleme ile oluşturulan test senaryosu.

4.2.5. Arama Operatörü

Yeni birey üretimi ve var olan bireyin değiştirilmesi olmak üzere iki temel arama operatörü bulunmaktadır. Yeni bir test oluşturulduğunda, Swagger şemasından elde edilen bilgiye göre uygun kromozom şablonu kullanılır ve 1 ile n arasında rastgele HTTP çağrısı oluşturulur. Genlerdeki değerler Swagger'dan (Örnek, Şekil 4.1) elde edilen min-max değerler arasından seçilmektedir.

Var olan bir testin değiştirilmesi için mutasyon operatörü kullanılmaktadır. Her bir test senaryosu, yapısı (HTTP çağrısı ekleme veya çıkarma gibi) veya değeri değiştirilerek mutasyona uğratılır. Her bir testin mutasyona uğrama olasılığı $1/k$ 'dır. Burada k HTTP çağrısındaki gen sayısını temsil etmektedir. Mutasyon operatörü aşağıda belirtildiği gibi

literatürdeki [143] örneklere benzer çalışmaktadır.

- Boole değeri tersine çevrilir.
- Tam sayılar $\pm 2^i$ delta ile değiştirilir. Buradaki "i" rastgele olarak 0 ile arama sırasında azalan bir maksimum değer arasından seçilmektedir.
- Float/double değerler için aynı tür $\pm 2^i$ delta uygulanır ancak gauss değeri ile çarpılmaktadır.
- Dizeler özel olarak ele alınmaktadır. Çünkü bunların en iyi nasıl değiştirileceği amaç fonksiyon değerine bağlıdır. Bir karakter rastgele değiştirilebilmekte, son karakter bırakılabilmekte veya dizinin sonuna yeni oluşturulan bir karakter eklenebilmektedir.

Örneğin Şekil 4.9'da verilen 7 HTTP isteğinden oluşan rastgele oluşturulmuş test senaryosu örneği incelendiğinde, bu test senaryosunda herhangi bir satır silinebilir, yeni HTTP isteği eklenebilir veya bunlardan biri değiştirilebilir durumdadır. *GET /products/SNlpoIbZ2Li* isteğinin değiştirildiği varsayıldığında, yeni test senaryosu Şekil 4.10'daki gibi olmaktadır.

```

1  POST /products/9Xs3n1XecXX1C2/constraints/excludes
2  GET /products/YJ
3  POST /products/WLiv4O2z/features/U
4  GET /products
5  DELETE /products/2D/constraints/28069080
6  GET /products/SNlpoIbZ2Li
7  GET /products/X/features

```

Şekil 4.9. Yedi HTTP isteğinden oluşan örnek test senaryosu.

```

1  POST /products/9Xs3n1XecXX1C2/constraints/excludes
2  GET /products/YJ
3  POST /products/WLiv4O2z/features/U
4  GET /products
5  DELETE /products/2D/constraints/28069080
6  GET /products/SNlpoIbZB%5Ci
7  GET /products/X/features

```

Şekil 4.10. Mutasyona uğramış test senaryosu.

Eğer test senaryosu akıllı örnekleme ile oluşturulduysa yapıyı değiştiren mutasyonların gerçekleşmesine izin verilmemektedir. Bunun sebebi herhangi bir çağrının kaldırılması

ile zincirleme konum çağrılarının bozulacak veya alakasız uç noktalara çağrı ekleyerek testin çalışmasında daha fazla efor sarf etmesine sebep olacak olmasıdır. Tek istisna GET şablonuna ait k parametresidir. Yani POST çağrılarını ekleyip kaldırmaya izin verilmektedir. Bir başka dikkate alınması gereken nokta ise zincirleme konum çağrılarında yol parametrelerinin değiştirilmesinin, bu parametreler dinamik olarak oluşturulduğu için bir anlamının olmamasıdır. Bu nedenle akıllı örnekleme ile oluşturulmuş test senaryolarında yalnızca yük, başlık ve sorgu parametrelerinin değiştirilmesine izin verilmektedir.

Bütün testler aynı amaç fonksiyonu ile değerlendirildiği için akıllı örnekleme yöntemi kötü senaryolar oluştursa da zaman içinde yok olacaktır.

4.3. RESTful Web Servisler için Test Senaryosu Üretiminde ABC Algoritması

Bölüm 3.4'te verilen Temel ABC algoritmasının gösterimi ve arama operatörlerinin, kombinasyonel problem olan test senaryo üretim problemine uygun olarak güncellenmesi gerekmektedir. Bu nedenle ABC algoritmasının gösterimi, arama operatörleri ve fazlarında değişiklikler gerçekleştirilerek RESTful test senaryosu üretimi için uygun hale getirilmiştir ve bu algoritmanın sözde kodları Algoritma 4.1'de verilmiştir. Görevli arı fazında yeni çözüm üretmek için rastgele bir çözüm belirlenmekte ve mevcut çözüm ile önce çaprazlama operatörü daha sonra mutasyon operatörüne tabi tutulmaktadır. Ardından aç gözlü seçim kullanılarak belirlenen en iyi çözüm mevcut çözümün yerini almaktadır. Herhangi bir iyileşme olmaz ise mevcut çözüm yerini korumakta ve ilgili sayaç (*trial*) değeri bir artırılmaktadır. Gözcü arı fazında ise arılar Eşitlik 4.1 ile hesaplanan p_i olasılığına bağlı olarak ilgili yiyecek kaynaklarının çevresinde arama eylemlerini gerçekleştirmektedir. p_i olasılığına bağlı olarak seçilen çözüm ile rastgele bir çözüme çaprazlama ve mutasyon operatörleri uygulanmakta ve oluşturulan çözümler arasında aç gözlü seçim ile popülasyonda kalacak birey belirlenmektedir. Herhangi bir iyileşme olmaması durumunda ilgili sayaç (*trial*) değeri bir artırılmaktadır. Eğer çözümlerden birinin ilgili sayaç (*trial*) değeri kontrol parametresi olan *limit*'i aşarsa bu yiyecek kaynağının yerini rastgele yeni bir kaynak alır. RESTful web servislerin testi için geliştirilen ABC algoritmasının popülasyon büyüklüğü (PS), maksimum değerlendirme sayısı (MFE), *limit*, çaprazlama olasılığı (CR), mutasyon olasılığı (MR) olmak üzere

Algoritma 4.1’de gösterildiği gibi çeşitli kontrol parametreleri bulunmaktadır.

$$p_i = \frac{fitness_i}{\sum_{j=1}^{CS} fitness_j} \quad (4.1)$$

```

1: { //Başlangıç aşaması }
2:  $PS \leftarrow$  Popülasyon büyüklüğü
3:  $MFE \leftarrow$  Maksimum değerlendirme sayısı
4:  $limit \leftarrow$  Tükenmiş kaynakları belirlemek için maksimum deneme sayısı
5:  $MR \leftarrow$  Mutasyon oranı
6:  $CR \leftarrow$  Çaprazlama olasılığı
7:  $c_f \leftarrow$  Çaprazlama fonksiyonu
8:  $t_i \leftarrow 0, i = 1 \dots PS$ 
9:  $\delta \leftarrow AmacFonksiyonu$ 
10:  $P \leftarrow RastgeleYiyecekKaynagiUret(PS)$ 
11:  $AmacFonksiyonu(\delta, P)$ 
12: while  $!maksimumDegerlendirmeSayisiMi(MFE)$  do
13:   { //Görevli An Fazı }
14:    $i \leftarrow 0$ 
15:   while  $i < PS$  ve  $!maksimumDegerlendirmeSayisiMi(MFE)$  do
16:      $p_1 \leftarrow P_i$ 
17:      $p_2 \leftarrow P_{rand}$ 
18:      $o_1, o_2 \leftarrow Caprazlama(c_f, CR, p_1, p_2)$ 
19:      $o_1 \leftarrow Mutasyon(o_1, MR)$ 
20:      $o_2 \leftarrow Mutasyon(o_2, MR)$ 
21:      $AmacFonksiyonu(\delta, o_1)$ 
22:      $AmacFonksiyonu(\delta, o_2)$ 
23:     if  $EnIyi(o_1, o_2), p_1$  'den daha iyiye then
24:        $p_1 \leftarrow EnIyi(o_1, o_2)$ 
25:     else
26:        $DenemeSayaciniArtir(p_1)$ 
27:     end if
28:      $i++$ 
29:   end while
30:   { //Gözcü An Fazı }
31:    $prob \leftarrow OlasilikHesapla(P)$ 
32:    $i \leftarrow 0$ 
33:    $k \leftarrow 0$ 
34:   while  $k < PS$  ve  $!maksimumDegerlendirmeSayisiMi(MFE)$  do
35:     if  $rand < prob_i$  then
36:        $k++$ 
37:        $p_1 \leftarrow P_i$ 
38:        $p_2 \leftarrow P_{rand}$ 
39:        $o_1, o_2 \leftarrow Caprazlama(c_f, CR, p_1, p_2)$ 
40:        $o_1 \leftarrow Mutasyon(o_1, MR)$ 
41:        $o_2 \leftarrow Mutasyon(o_2, MR)$ 
42:        $AmacFonksiyonu(\delta, o_1)$ 
43:        $AmacFonksiyonu(\delta, o_2)$ 
44:       if  $EnIyi(o_1, o_2), p_1$  'den daha iyiye then
45:          $p_1 \leftarrow EnIyi(o_1, o_2)$ 
46:       else
47:          $DenemeSayaciniArtir(p_1)$ 
48:       end if
49:     end if
50:      $i \leftarrow ++i \bmod PS$ 
51:   end while
52:   { //Kâşif An Fazı }
53:    $i \leftarrow 0$ 
54:   while  $i < PS$  AND  $!maksimumDegerlendirmeSayisiMi(MFE)$  do
55:     if  $t_i > limit$  then
56:        $P_i \leftarrow YeniBirey()$ 
57:     end if
58:      $i++$ 
59:   end while
60:    $UygunlukDegerleriniGuncelle(P, \delta)$ 
61: end while

```

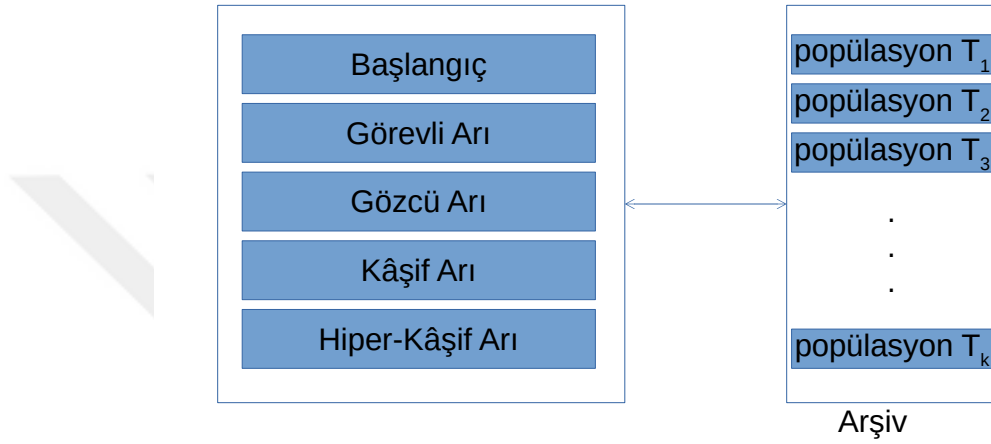
Algoritma 4.1: RESTful Test senaryo üretimin için ABC algoritmasının sözde kodları

4.4. Hiper-Kâşif Arı Fazlı Dinamik Yapay Arı Koloni Algoritması (DABC-HS)

Gerçekleştirilen ilk deneyler sonucunda, hesaplama zamanının çoğunun ABC algoritmasının faydalanma mekanizmaları tarafından tüketildiği için temel ABC algoritmasının RESTful test senaryosu oluşturmadaki performansının tatmin edici olmadığını görülmüştür. Özellikle hesaplama zamanı sınırlı olduğunda, ABC algoritmasının sağladığı keşif yeterli olamamıştır. ABC algoritmasının keşif mekanizmasının iyileştirilmesi gerektiği için popülasyonun yakınsamasına katkının sağlanmadığı durumlarda devreye giren Hiper-Kâşif arı fazı eklenerek popülasyonda yüksek çeşitlilik sağlanmıştır.

Temel algoritmanın bir başka dezavantajı ise optimize edilmiş bireylerin henüz kapsanmamış hedeflerle herhangi bir bağı olmasa da popülasyonda varlığını sürdürüyor olmasıdır. Bu problemin üstesinden gelebilmek için gözcü arı fazına bir havuz mekanizması eklenmiştir. Bu fazda üretilen bütün bireyler bu havuza atılır ve baskınlığa dayalı bir seçim mekanizması kullanılarak, popülasyon uygun sayıya tekrar düşürülür. Böylece optimize edilmiş bireylerin popülasyonda varlığını sürdürmesi engellenmektedir. Ayrıca önerilen bu yöntem her bir hedef için en iyi bireyleri (optimize bireyler olması şart değil) arşivde tutmaktadır. Gözcü arı fazı da belli bir periyotta arşivden örneklemeler yaparak performansı artırmayı hedeflemektedir. Temel ABC algoritmasında yerel arama sonucunda bir iyileşme sağlanmaz ise çözümün sayacı bir (1) artırılmaktadır. Bu sayaç değerine göre de kâşif arı fazında ilgili kaynak bırakılmakta ve rastgele yeni bir kaynak popülasyona katılmaktadır. Ancak önerilen yöntemde, temel ABC'nin aksine bu sayaçlar sadece kâşif arı fazında değil, tercihli seçim kısmında da kullanılmaktadır. Arşivden örneklemeler düşük sayaca sahip bireyler arasından seçilmektedir. Böylece düşük kaliteli, çözümü olmayan yollar içeren bireylerin örneklenmesine izin verilmeyerek yüksek kaliteli çözümlerin popülasyonda ve arşivde var olma ihtimali artmaktadır. Hedeflerin sayısı tüm arama boyunca yeni hedef keşfedildikçe değiştiği için görevli ve gözcü arı fazlarına aynı hesaplama kaynağının ayrılması bu problem için çok da uygun olmamaktadır. Bu nedenle önerilen yöntemde, dinamik olarak, görevli arı ve gözcü arı fazına ayrılan kaynaklar değişmektedir. RESTful web servisler için test senaryosu

üretimine uygun olarak önerilen yöntem, belirtilen özellikleri sebebiyle, Hiper Kâşif Fazlı Dinamik Yapay Arı Koloni Algoritması (DABC-HS) adı verilmiştir. Şekil 4.11’de DABC-HS algoritmasının genel yapısı verilmiştir. Sonraki alt bölümlerde Şekil 4.11’deki bloklar açıklanmıştır. DABC-HS algoritmasının sözde kodları ise Algoritma 4.2’de verilmiştir.



Şekil 4.11. DABC-HS algoritmasının genel yapısı.

4.4.1. Arşiv Bloğu

Arama sırasında en iyi çözümlerin popülasyonda tutulması çeşitliliği azaltmaktadır. Daha önce söz edildiği gibi çeşitliliğin korunması için kâşif arı fazında bireyler çeşitli aralıklarla değiştirilmektedir. Ancak çeşitlilik yeterli seviye ulaştığında da arama üzerinde yoğunlaşabilmek için bu çözümlere ihtiyaç duyulabilmektedir. Ayrıca kullanıcıya verilmek üzere son test paketi olarak oluşturulması için en az bir hedefi kapsayan test senaryolarına ihtiyaç vardır. Bu nedenlerden dolayı önerilen algoritmada, Şekil 4.12’de olduğu gibi her bir hedefe ait en iyi test senaryolarının tutulduğu bir arşiv modeli kullanılmaktadır.

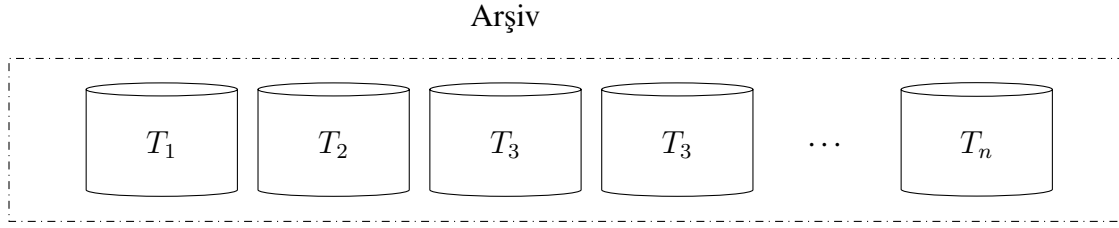
Aramanın başında arşiv boş olmaktadır. z hedefe sahip bir SUT test edilmeye başlandığında, z değişken boyutlu popülasyona sahip arşiv $(T_k, (k = 1...z))$ oluşturulmaktadır. Algoritma ne zaman yeni bir hedef keşfetse veya bir hedefi iyileştirecek olsa, bir veya daha fazla hedefe bu test senaryosu eklenmektedir. Her bir

```

1: {Başlangıç aşaması}
2:  $PS \leftarrow$  Popülasyon büyüklüğü
3:  $MFE \leftarrow$  Maksimum değerlendirme sayısı
4:  $SizeFactor \leftarrow \{sf_1, sf_2, \dots, sf_{cn}\}$ , cn farklı ayar sayısıdır
5:  $ChangeNumber \leftarrow$   $CurrentSizeFactor$  değerini değiştirmek için gerekli maksimum iyileştirme olmayan iterasyon sayacı
6:  $ResetNumber \leftarrow$  Popülasyonu tekrar başlatmak için gerekli maksimum iyileştirme olmayan iterasyon sayacı
7:  $limit \leftarrow$  Tükümlü kaynakları belirlemek için maksimum deneme sayısı
8:  $MR \leftarrow$  Mutasyon oranı
9:  $P_r \leftarrow$  Akıllı örnekleme oranı
10:  $threshold \leftarrow$  Maksimum sayıda mutasyonun uygulanması için gerekli eşik değeri
11:  $n \leftarrow$  Arşivdeki her bir popülasyonun üst limiti
12:  $t_i \leftarrow 0, i = 1 \dots PS$ 
13:  $SizeFactorCounter \leftarrow 0$ 
14:  $CurrentSizeFactor \leftarrow SizeFactor[SizeFactorCounter]$ 
15:  $arsiv \leftarrow$   $arsivBaslat(n)$ 
16:  $P \leftarrow RastgeleYiyecekKaynagiUret(PS * CurrentSizeFactor)$ 
17:  $ni \leftarrow 0$  //iyileştirme görülmeyen iterasyon sayacı
18: while  $!maksimumDegerlendirmeSayisiMi(MFE)$  do
19:   while  $size(P) < PS * CurrentSizeFactor$  ve  $!maksimumDegerlendirmeSayisiMi()$  do
20:      $P \leftarrow P \cup bireyOrnekle(arsiv, P_r)$ 
21:   end while
22:   {Görevli Arı Fazı}
23:   for  $i < PS$  do
24:      $yeniBirey \leftarrow mutasyon(mutasyonSayisiniAl(threshold), P_i, arsiv, MR)$ 
25:     if  $yeniBirey, P_i$ 'den daha iyiyse then
26:        $P_i \leftarrow yeniBirey$ 
27:        $t_i \leftarrow 0$ 
28:     else
29:        $t_i \leftarrow t_i + 1$ 
30:     end if
31:   end for
32:   {Gözcü Arı Fazı}
33:    $sayac \leftarrow 0$ 
34:   while  $sayac < PS - PS * CurrentSizeFactor$  ve  $!maksimumDegerlendirmeSayisiMi(MFE)$  do
35:      $yeniBirey \leftarrow$   $arsivdenOrneklemeVeMutasyon(arsiv, MR, threshold)$ 
36:     if  $yeniBireyEklendi$  then
37:        $P \leftarrow P \cup yeniBirey$ 
38:        $t_i \leftarrow 0$ 
39:     end if
40:      $t_i \leftarrow t_i + 1$ 
41:      $sayac \leftarrow sayac + 1$ 
42:   end while
43:    $\mathbb{F} \leftarrow TercihliSecim(P)$ 
44:    $P \leftarrow PopulasyonIndirgemesi(\mathbb{F}, PS)$ 
45:   {Kâşif Arı Fazı}
46:   for  $i < PS$  do
47:     if  $t_i > limit$  ve  $!maksimumDegerlendirmeSayisiMi(MFE)$  then
48:        $P_i \leftarrow bireyOrnekle(arsiv, P_r)$ 
49:        $t_i \leftarrow 0$ 
50:     end if
51:   end for
52:   {Hiper-Kâşif Arı Fazı ve Parametre Değişimi}
53:   if iyileşme gerçekleşmediyse then
54:      $ni \leftarrow ni + 1$ 
55:     if  $ni > ChangeNumber$  then
56:        $CurrentSizeFactor \leftarrow SizeFactor[+ + sizeFactorCounter \bmod cn]$ 
57:     end if
58:     if  $ni > ResetNumber$  then
59:        $P \leftarrow RastgeleYiyecekKaynagiUret(PS * CurrentSizeFactor)$ 
60:        $ni \leftarrow 0$ 
61:     end if
62:   else
63:      $ni \leftarrow 0$ 
64:   end if
65: end while

```

Algoritma 4.2: DABC-HS algoritmasının sözde kodları



Şekil 4.12. Arşiv mekanizmasının genel yapısı.

T_i popülasyonu n test sayısı ile sınırlandırılmıştır ($|T_k| \leq n$). Böylece maksimum $n \times z$ adet test arşivde bulunabilmektedir. Her bir hedefe $[0, 1]$ aralığında sezgisel değer (h) atanmaktadır. 0 değeri kapsanmamış hedef için en kötü değer olurken, 1 ise ilgili hedefin kapsandığını göstermektedir. k hedefi için h_k değeri Eşitlik 4.2 ile hesaplanmaktadır. Burada d_k değeri dal mesafesini [57] ifade etmektedir.

Eğer bir dala hiç erişilemez ise ilgili h_k değeri hesaplanamaz [158] ve bu durumda h_k değeri 0 olarak atanır. Her bir k hedefi için aşağıdaki şartlar sağlandığında arşiv popülasyonuna eklenmektedir.

- Eğer $h_k = 0$ ise herhangi bir testin arşiv popülasyonu T_k 'ye eklenmesine izin verilmez.
- Eğer $h_k = 1$ ise test T_k popülasyonuna eklenebilir ve ilgili popülasyon tek bir birey olacak şekilde küçültülür ve tekrar büyümesine izin verilmez. Eğer yeni test T_k popülasyonundaki testten daha küçük ve aynı kapsama miktarına sahipse veya aynı boyutta fakat farklı hedefler için daha iyi kapsama değerine sahipse değiştirilmektedir.
- Eğer $0 < h_k < 1$ ve $|T_k| < n$ ise T_k popülasyonuna test eklenmektedir. $|T_k| = n$ olduğunda ise yeni ve daha iyi test en kötü test ile değiştirilmektedir.

Eğer k hedefi kapsanırsa, en iyi test senaryosu arşiv popülasyonunda tutulmakta ve gerisi popülasyondan silinmektedir. Ancak çıkarılan test diğer hedefler için hâlâ iyi bir test ise diğer arşiv popülasyonunda tutulmaya devam edilmektedir.

$$h_k = 1/(1 + d_k) \quad (4.2)$$

4.4.2. Başlangıç

DABC-HS algoritmasında, temel ABC'nin kontrol parametrelerine (PS , MFE , $limit$, MR) ek olarak $SizeFactor$, $ChangeNumber$ ve $ResetNumber$ olmak üzere üç yeni kontrol parametresi bulunmaktadır. $SizeFactor$ görevli arı fazındaki popülasyon sayısını belirlemektedir. $PS * SizeFactor$ kadar birey görevli arı fazının popülasyon sayısı olurken, arta kalan popülasyon da gözcü arı fazının popülasyonu olmaktadır. $ChangeNumber$ sayısı kadar iterasyon içerisinde popülasyonda herhangi bir iyileşme meydana gelmez ise $SizeFactor$ parametresi değişmektedir. Benzer şekilde, $ResetNumber$ sayısı kadar iterasyonda popülasyonda bir iyileşme gerçekleşmez ise mevcut popülasyon tamamen atılmakta ve başlangıç aşamasında olduğu gibi yeni popülasyon üretilmektedir. Bu parametrelerin değerleri de başlangıç aşamasında atanmaktadır. Başlangıç popülasyonu belli bir P_r olasılığına bağlı olarak akıllı örnekleme [143] ile oluşturulmaktadır.

4.4.3. Görevli Arı Fazı

Daha önce belirtildiği gibi DABC-HS algoritmasında görevli ve gözcü arı sayıları dinamik olarak değişmektedir. $[0, 1]$ aralığında değer alan $SizeFactor$ kontrol parametresi kullanılarak $PS * SizeFactor$ tane birey görevli arı olarak belirlenmiştir. Keşif ve faydalanma yeteneğini artırmak için $SizeFactor$ kontrol parametresi dinamik olarak değişmektedir. Algoritma 4.2, satır 19-21'de görüldüğü gibi eğer popülasyon büyüklüğü $PS * SizeFactor$ 'den küçük ise yeni bireyler eklenerek popülasyon uygun boyuta getirilmektedir. Görevli arı fazında uygulanan mutasyonun sayısı da adaptif olarak değişmekte ve mutasyon sayısı Eşitlik 4.3 ile hesaplanmaktadır.

$$MutasyonSayisi = \begin{cases} max, & \text{Eğer } usedBudget \geq threshold. \\ min + \frac{max - min}{usedBudget/threshold}, & \text{aksi takdirde.} \end{cases} \quad (4.3)$$

Burada $usedBudget$ o ana kadarki harcanan HTTP çağrı sayısını, $threshold$ da

maksimum sayıda mutasyon uygulanması için gerekli HTTP çağrı sayısını vermektedir.

Temel ABC algoritmasında söz edildiği gibi her bir çözümün bir sayacı bulunmaktadır. Mutasyona uğrayan birey daha iyi ise bu sayaç sıfırlanırken, eğer bir iyileşme gerçekleşmezse bu sayaç bir (1) artırılmaktadır.

4.4.4. Gözcü Arı Fazı

Gözcü arı fazında görev alan gözcü arı sayısı $PS \cdot (1 - SizeFactor)$ kadardır. Bu aşamada yeni çözüm üretilirken, DABC-HS arşivden birey örneklemekte ve bu örneklenen bireye mutasyon operatörü uygulamaktadır. Bazı çok amaçlı problemlerde hedef kapsanabilir olsa da tamamını kapsayacak kadar kaynak bulunamayabilir. Son kullanıcı, hedefler kapsamaya yakın olsa da bununla değil yalnızca kapsanmış hedeflerle ilgilendiğinden daha yüksek kapsama ihtimali olan hedeflere odaklanmak daha mantıklı bir çözümdür. Bu nedenle DABC-HS algoritmasında, [158]'da Arcuri tarafından önerilen basit ve etkin bir yöntem kullanılmıştır. Arşiv popülasyonu olan T_k 'ye c_k sayacı atanmaktadır ve her bir arşivden örneklemeye gerçekleştiğinde değeri bir (1) artırılmaktadır. Eğer T_k popülasyonuna yeni ve daha iyi bir test eklenirse veya herhangi bir test daha iyi test ile değiştirilirse c_k değeri tekrar sıfır (0) olmaktadır. Arşivden bir çözüm örneklenmek istendiğinde, en düşük c_k değerine sahip arşiv popülasyonundan örneklemeye sağlanmaktadır. Eğer ilgili hedef kapsanma imkanı olmayan bir hedefse c_k değeri artacak ve bir noktadan sonra bir daha örneklemeye gerçekleşmeyecektir. Bu da kapsanabilir hedeflere odaklanılmasını ve kapsanan hedef sayısının artırılmasını sağlamaktadır. Ayrıca daha kolay hedeflerin daha önce kapsanmasına yardımcı olmaktadır.

Mutasyon sonucu oluşan yeni çözüm, herhangi bir hedef için mevcut çözümden daha iyiye hem arşive hem de popülasyona eklenmektedir ve popülasyonunun boyutu belirlenenden daha büyük hale gelmektedir. Bu nedenle, [179] çalışmasında olduğu gibi baskınlık tabanlı bir yöntem (Algoritma 4.3) kullanılarak tekrar popülasyon uygun boyuta getirilmektedir. TercihliSecim en düşük amaç skoruna sahip kapsanmamış hedefi ($u_i \in U$) belirlemektedir. Bu algoritma ile sonraki iterasyonlara sıralaması 0 olarak atanmış en düşük amaç skoruna sahip bireylerin kalma şansı artmaktadır. Sıralaması 0 olmayan test senaryoları ise geleneksel non-dominated yöntemi (Algoritma 4.5) ile sıralanmaktadır. Bu

sürecin daha hızlı olmasını sağlamak açısından, test sayısı popülasyon sayısından daha az olduğu zaman ilk baskılanamayan (non-dominated) front ($\mathbb{F}_0$) olarak adlandırılan test grubu (Algoritma 4.3'ün 8-9. satırları) kullanılmaktadır. $|\mathbb{F}_0| > PS$ sağlandığı zaman, sonraki sıralama değerine sahip test grubu olan $\mathbb{F}_1$ herhangi bir sıralamaya tabi olmadan arta kalan diğer test senaryolarından oluşmaktadır.

Popülasyon indirgemesinin temel adımları Algoritma 4.4'te [179] verilmiştir. En düşük sıralama değerine sahip bireyler istenen sayıya ulaşana kadar yeni popülasyona (Algoritma 4.4, satır 4-8) eklenmektedir. Eğer aynı sıralama değerine sahip bütün bireyler popülasyona eklenemez ise ($|P_{yeni}| + |\mathbb{F}_d| > PS$) yığılma uzaklığı değerine göre bireyler sıralanmakta ve en iyi $PS - |P_{yeni}|$ birey popülasyona eklenmektedir (Algoritma 4.4, satır 9-10).

Giriş: T : Aday test senaryoları, PS : Popülasyon boyutu

Çıktı: $\mathbb{F}$: Baskılanamayan sıralama atamaları

```

1:  $\mathbb{F}_0$  //ilk baskılanamayanlar
2: for  $u_i \in U$  ve  $u_i$  kapsanamamışsa do
3:   {/her bir kapsanamayan hedef için, en iyi test senaryosu seçimi}
4:    $t_{EnIyi} \leftarrow T$  içindeki,  $u_i$  için en düşük amaç fonksiyonu değerine sahip test senaryosu
5:    $\mathbb{F}_0 \leftarrow \mathbb{F}_0 \cup \{t_{EnIyi}\}$ 
6: end for
7:  $T \leftarrow T - \mathbb{F}_0$ 
8: if  $|\mathbb{F}_0| > PS$  then
9:    $\mathbb{F}_1 \leftarrow T$ 
10: else
11:    $U^* \leftarrow \{g \in U : u\}$  kapsanamamış
12:    $\mathbb{E} \leftarrow HizliBaskılanamayanSiralama(T, \{u \in U^*\})$ 
13:    $d \leftarrow 0$  //  $\mathbb{E}$  içindeki ilk frontlar
14:   for  $\mathbb{E}$ 'deki bütün baskılanamayan frontlar do
15:      $\mathbb{F}_{d+1} \leftarrow \mathbb{E}_d$ 
16:   end for
17: end if

```

Algoritma 4.3: TercihliSecim fonksiyonunun sözde kodları.

Input: PS : İndirgenecek popülasyon sayısı, $\mathbb{F}$: Non-dominated Front

Output: P_{yeni} : DABC-HS algoritmasının yeni popülasyonu

```

1:  $P_{yeni} \leftarrow \emptyset$ 
2:  $d \leftarrow 0$ 
3: while  $|P_{yeni}| + |\mathbb{F}_d| < PS$  do
4:    $YigilmaUzakligiAtamasi(\mathbb{F}_d)$ 
5:    $P_{yeni} \leftarrow P_{yeni} \cup \mathbb{F}_d$ 
6:    $d \leftarrow d + 1$ 
7: end while
8:  $Sirala(\mathbb{F}_d)$  //yığılma uzaklığına göre
9:  $P_{yeni} \leftarrow P_{yeni} \cup \mathbb{F}_d[1 : (PS - |P_{yeni}|)]$ 
10: return  $P_{yeni}$ 

```

Algoritma 4.4: Popülasyon indirgemesinin sözde kodları.

Giriş: t_1 ve t_2 : Karşılaştırılacak iki test senaryosu $U = \{u_1, \dots, u_m\}$: Kapsama hedefleri

```

1:  $dominates_1 \leftarrow false$ 
2:  $dominates_2 \leftarrow false$ 
3: for  $u_i \in U$  ve  $u_i$  kapsanmadıysa do
4:    $f_i^1 \leftarrow t_1$  için  $u_i$  değeri
5:    $f_i^2 \leftarrow t_2$  için  $u_i$  değeri
6:   if  $f_i^1 < f_i^2$  then
7:      $dominates_1 \leftarrow true$ 
8:   end if
9:   if  $f_i^2 < f_i^1$  then
10:     $dominates_2 \leftarrow true$ 
11:  end if
12:  if  $dominates_1 == true$  and  $dominates_2 == true$  then
13:    break
14:  end if
15: end for
16: if  $dominates_1 == dominates_2$  then
17:    $\{t_1$  ve  $t_2$  birbirlerini baskılayamamaktadır}
18: else
19:   if  $dominates_1 == true$  then
20:     $\{t_1, t_2$ 'yi baskılar}
21:   else
22:     $\{t_2, t_1$ 'i baskılar}
23:   end if
24: end if

```

Algoritma 4.5: Baskınlık karşılaştırma fonksiyonunun sözde kodları.**4.4.5. Kâşif Arı Fazı**

Kâşif arı fazında, ilgili sayaç (*trial*) değeri eğer belirlenen *limit* değerini aşarsa ilgili kaynak bırakılmaktadır ve yerini rastgele yeni bir kaynak almaktadır.

4.4.6. Hiper-Kâşif Arı Fazı

Hiper-kâşif arı fazı, kontrol parametre değişimi ve popülasyon yenilemesi olmak üzere iki temel aşamadan oluşmaktadır. Eğer *ChangeNumber* sayısı kadar iterasyonda iyileşme olmaz ise *SizeFactor* kontrol parametresi, Algoritma 4.2 satır 56'da olduğu gibi değişmektedir. Eğer *ResetNumber* sayısı kadar iterasyonda iyileşme olmaz ise mevcut popülasyon tamamen silinmekte ve yeni hedefler keşfetmek üzere popülasyon tekrar başlatılmaktadır.

4.5. Literatürde Bulunan Algoritmalar

RESTful web servisler için test senaryosu üretimi için literatürde en çok kullanılan yöntemlerden olan WTS, MOSA ve MIO algoritmalarının temelleri bu bölümde verilmiştir.

4.5.1. Tam Test Paketi

Tam Test Paketi (Whole Test Suite, WTS) Arcuri [56] tarafından EvoMaster³ test senaryosu üretim aracı için oluşturulmuş, SGA tabanlı bir algoritmadır. Yeni test senaryosu örneklendiğinde veya bu test senaryosunun komşu arama mekanizmalarında yeni hedef kapsadığı zaman çözümler arşive eklenmektedir.

4.5.2. Çok Hedefli Sıralama Algoritması

Çok Hedefli Sıralama Algoritması (Many-Objective Sorting Algorithm, MOSA) [179] WTS'in kısıtlamalarını gidermek için ortaya çıkmış test senaryosu üretimi için kullanılan algoritmalarından biridir. MOSA'da her bir amaç optimize edilmesi gereken bir hedef olarak modellenmiştir. MOSA, çok amaçlı optimizasyon problemlerinde oldukça popüler olan NSGA-II [184] algoritmasını temel almaktadır. MOSA'da herhangi bir hedef kapsadığı zaman arşive eklenmekte ve amaç fonksiyonu hesaplamasında kullanılmamaktadır. Arama sırasında elde edilen en iyi bireylerin arşivde saklanması ile oluşan en iyi sonuçlar kümesi optimum çözüm kümesidir.

³<https://github.com/EMResearch/EvoMaster>

Giriş: C : Durdurma kriteri, $U = \{u_1, \dots, u_m\}$: Kapsama hedefleri, PS : Popülasyon boyutu
Çıktı: T : Test paketi

```

1:  $t \leftarrow 0$  // ilk iterasyon
2:  $P_t \leftarrow RastgelePopulasyon(PS)$ 
3:  $arsiv \leftarrow ArsivGuncelle(P_t, \emptyset)$ 
4: while  $\neg C$  do
5:    $Q_t \leftarrow CocukCozumUret(P_t)$ 
6:    $arsiv \leftarrow ArsivGuncelle(Q_t, arsiv)$ 
7:    $R_t \leftarrow P_t \cup Q_t$ 
8:    $\mathbb{F} \leftarrow TercihliSecim(R_t)$ 
9:    $P_{t+1} \leftarrow \emptyset$ 
10:   $d \leftarrow 0$ 
11:  while  $|P_{t+1}| + |\mathbb{F}_d| \leq PS$  do
12:     $CrowdingDistance(\mathbb{F}_d)$ 
13:     $P_{t+1} \leftarrow P_{t+1} \cup \mathbb{F}_d$ 
14:     $d \leftarrow d + 1$ 
15:  end while
16:   $Sirala(\mathbb{F}_d)$  //yığılma uzaklığına göre
17:   $P_{t+1} \leftarrow P_{t+1} \cup \mathbb{F}_d[1 : (PS - |P_{t+1}|)]$ 
18:   $t \leftarrow t + 1$ 
19: end while
20:  $T \leftarrow arsiv$ 

```

Algoritma 4.6: MOSA algoritmasının sözde kodları

Giriş: T : Aday test senaryoları, PS : Popülasyon boyutu
Çıktı: $\mathbb{F}$: Baskılanamayan sıralama atamaları

```

1:  $\mathbb{F}_0$  //ilk baskılanamayanlar
2: for  $u_i \in U$  ve  $u_i$  kapsanmamışsa do
3:   {/her bir kapsanamayan hedef için, en iyi test senaryosu seçimi}
4:    $t_{EnIyi} \leftarrow T$  içindeki,  $u_i$  için en düşük amaç fonksiyonu değerine sahip test senaryosu
5:    $\mathbb{F}_0 \leftarrow \mathbb{F}_0 \cup \{t_{EnIyi}\}$ 
6: end for
7:  $T \leftarrow T - \mathbb{F}_0$ 
8: if  $|\mathbb{F}_0| > PS$  then
9:    $\mathbb{F}_1 \leftarrow T$ 
10: else
11:    $U^* \leftarrow \{g \in U : u\}$  kapsanmamış
12:    $\mathbb{E} \leftarrow HizliBaskılanamayanSiralama(T, \{u \in U^*\})$ 
13:    $d \leftarrow 0$  //  $\mathbb{E}$  içindeki ilk frontlar
14:   for  $\mathbb{E}$ 'deki bütün baskılanamayan frontlar do
15:      $\mathbb{F}_{d+1} \leftarrow \mathbb{E}_d$ 
16:   end for
17: end if

```

Algoritma 4.7: TercihliSecim fonksiyonunun sözde kodları.

MOSA'nın temel adımları Algoritma 4.6'da verilmiştir [185]. İlk olarak PS boyutunda rastgele bir popülasyon üretilmektedir ve içlerinde hedef kapsayan çözümler arşive eklenmektedir (Satır 2-3). Ardından durdurma kriteri sağlanana kadar tekrarlı olarak gerçekleştirilecek eylemler yürütülmektedir. İlk olarak çaprazlama ve mutasyon operatörleri kullanılarak Q_t çocuk çözümü üretilir (Satır 5). Bu çözüm herhangi bir hedefi kapsarsa veya kapsanmış bir hedef için hem kapsayıp hem de daha kısa olması durumunda arşive eklenmektedir. Daha sonra yeni üretilen bu birey ve popülasyondaki bireyler (R_t)

Algoritma 4.7’de kaba kodları verilen Tercihli Seçim işlemine tabi olurlar (Satır 8). Eğer yeterli popülasyona erişilemez ise baskılanamayan testler kendi içinde yığılma uzaklığına (crowding distance) göre sıralanır (Satır 11-16). Daha sonra bu sıralamaya göre en iyi değerler popülasyona katılmaktadır (Satır 17). Arama sonunda ise arşivdeki bireyler son test paketi olarak kullanıcıya sunulmaktadır.

Giriş:

t_1 ve t_2 : Karşılaştırılacak iki test senaryosu

$U = \{u_1, \dots, u_m\}$: Kapsama hedefleri

```

1:  $dominates_1 \leftarrow false$ 
2:  $dominates_2 \leftarrow false$ 
3: for  $u_i \in U$  ve  $u_i$  kapsanmadıysa do
4:    $f_i^1 \leftarrow t_1$  için  $u_i$  değeri
5:    $f_i^2 \leftarrow t_2$  için  $u_i$  değeri
6:   if  $f_i^1 < f_i^2$  then
7:      $dominates_1 \leftarrow true$ 
8:   end if
9:   if  $f_i^2 < f_i^1$  then
10:     $dominates_2 \leftarrow true$ 
11:  end if
12:  if  $dominates_1 == true$  and  $dominates_2 == true$  then
13:    break
14:  end if
15: end for
16: if  $dominates_1 == dominates_2$  then
17:    $\{t_1$  ve  $t_2$  birbirlerini baskılayamamaktadır}
18: else
19:   if  $dominates_1 == true$  then
20:     $\{t_1, t_2$ ’yi baskılar}
21:   else
22:     $\{t_2, t_1$ ’i baskılar}
23:   end if
24: end if

```

Algoritma 4.8: Baskınlık karşılaştırma fonksiyonunun sözde kodları.

Algoritma 4.7’de verilen Tercihli Seçim işleminde [185], her kapsanamayan hedef için en iyi test senaryolarını Sıralama 0 olarak ($\mathbb{F}_0$) atamaktadır (Satır 2-6). Eğer $\mathbb{F}_0$ ’daki birey sayısı PS sayısından büyük ise artakalan testler Sıralama 1 olarak ($\mathbb{F}_1$) atanmaktadır (Satır 9). Eğer büyük değilse geri kalan testler Deb [184] tarafından önerilen Hızlı Baskılanamayan Sıralama (Fast Non-Dominated Sort) ile Sıralama değerleri belirlenmektedir. Bu aşamada bireyler, baskın olma dereceleri dikkate alınarak farklı baskınlık kümelerine göre ayrılmaktadır. İki testin birbirine baskın olup olmadığı Algoritma 4.8’de verilen adımlarla belirlenmektedir [185].

4.5.3. Çoklu Bağımsız Hedef Algoritması

Çoklu Bağımsız Hedef Algoritması (Many Independent Objective, MIO) 2017 yılında Andrea Arcuri tarafından test senaryosu üretimi için önerilmiştir [157]. Algoritmanın çözmeyi hedeflediği problemler aşağıda belirtilmiştir.

- Popülasyon tabanlı algoritmalar keşif aşamasının üzerinde daha fazla durmaktadır. Bu da kısıtlı sistem gereksinimi olduğu durumlarda yüksek hesaplama maliyeti nedeniyle sorunlara yol açmaktadır.
- Herhangi bir hedef kapsandıktan sonra amaç fonksiyonunda hesaplamaya dahil edilmese bile, optimize edilmiş bireyler popülasyonda varlığını sürdürmeye devam etmektedir. Birkaç iterasyon sonra popülasyonda yok olacaktır fakat o zamana kadar bu bireyler henüz kapsanmamış hedeflerle bir bağlantısı yoksa arama sürecine yük olmaktadır.
- Erişilemez hedeflerin varlığı, bazı hedeflerin asla kapsanamayacak olsa da popülasyonda uygunluk değeri yüksek bireylerin var olmasına sebep olmaktadır. Bir süre sonra bu tür bireyler popülasyonun büyük bir çoğunluğunu oluşturabilmektedir.
- Kapsanması gereken hedefler oldukça fazla sayıda olabilmektedir. Sabit boyutlu popülasyon bu nedenle ilgili problemin çözümü için uygun olmayabilir. Eğer popülasyon boyutu çok küçük olursa çeşitlilik sağlanamayacak, çok büyük olursa da yakınsama hızı düşecektir. Ayrıca hesaplama maliyeti de oldukça fazla artacaktır.

Bu problemleri çözmek için MIO algoritması önerilmiştir. (1+1)-EA algoritmasının basitliğini temel almakla beraber dinamik popülasyon boyutu, dinamik keşif/faydalanma dengesi ve geri bildirimli yönlendirme hedef seçim mekanizması geliştirilmiştir.

MIO algoritmasının temel adımları Algoritma 4.9’da verilmiştir.

Aramanın başında boş bir popülasyon ve arşiv oluşturulmaktadır. Ardından P_r rastgele örnekleme olasılığına bağlı olarak rastgele bir birey üretilmektedir. Eğer birey rastgele üretilmediyse mutasyon operatörü uygulanmaktadır (Satır 7-8). Ardından kapsamış

olduğu hedefler için ilgili birey arşive eklenir ve popülasyondan çıkarılır (Satır 12-13). Eğer hedef kapsamadıysa bu birey popülasyona eklenmektedir. Eğer popülasyon büyüklüğü, maksimum popülasyon büyüklüğü n 'den büyük olursa en kötü amaç fonksiyonu değerine sahip test silinmektedir (Satır 17). İterasyon sonunda ise adaptif parametreler güncellenmektedir. Bir testin uygunluk değeri hesaplandığında, arşivdeki

Giriş: C : Durdurma kriteri, δ : Amaç fonksiyonu, n : Maksimum popülasyon büyüklüğü, m_f : Mutasyon fonksiyonu, m_p : Mutasyon oranı, P_r : Rastgele örnekleme olasılığı, F : Odaklanmış arama başlaması

Çıktı: A : Optimize edilmiş arşiv popülasyonu

```

1:  $Z \leftarrow \text{BosPopulasyonOlustur}()$ 
2:  $A \leftarrow \{\}$ 
3: while  $\neg C$  do
4:   if  $P_r > \text{Rastgele}(0, 1)$  then
5:      $p \leftarrow \text{RastgeleBireyUret}()$ 
6:   else
7:      $p \leftarrow \text{BireyOrnekle}(Z)$ 
8:      $p \leftarrow \text{Mutasyon}(m_f, m_p, p)$ 
9:   end if
10:  for each  $t \in \text{UlasilanHedefler}(p)$  do
11:    if  $\text{HedefKapsandiMi}(t)$  then
12:       $\text{ArxivGuncelle}(A, p)$ 
13:       $Z \leftarrow Z \setminus \{Z_t\}$ 
14:    else
15:       $Z \leftarrow Z_t \cup \{p\}$ 
16:      if  $|Z_t| > n$  then
17:         $\text{EnKotuTestiSil}(Z_t, \delta)$ 
18:      end if
19:    end if
20:  end for
21:   $\text{ParametreleriGuncelle}(F, P_r, n)$ 
22: end while
23: return  $A$ 

```

Algoritma 4.9: MIO algoritmasının sözde kodları

popülasyonun birden fazlasına eklenebilmektedir. Her oluşturulan hedef popülasyona $|T_k| \leq n$ olmak üzere test eklenebilir. Her bir hedefin $[0, 1]$ aralığında olmak üzere h_k sezgisel değeri bulunmaktadır. Bu sezgisel değer;

- $h_k = 0$ ise test arşive eklenmemektedir.
- $h_k = 1$ ise test arşive eklenmektedir ve arşiv popülasyonu yalnızca bu testi barındıracak şekilde 1'e düşürülmektedir. Aynı değeri sağlayan başka testler gelirse bu testlerin kısa olanı tercih edilmektedir. Eğer boyut da eşit ise toplam sezgisel değeri en yüksek olan test (diğer hedeflerde en fazla kapsama sağlayan) ile değiştirilmektedir.

- Eğer popülasyon dolu değilse ($T_k \leq n$) test popülasyona eklenmektedir. Eğer doluyorsa en kötü test ile değiştirilmektedir.

Keşif/faydalanma arasındaki denge ise P_r ve n değerlerinin zaman içerisinde değiştirilmesi ile sağlanmaktadır. Geri bildirimli yönlendirilmiş hedef seçimi ise; her bir seçilen birey için tutulan c_k değeri bir artırılmaktadır. Eğer seçildikten sonra iyileşme sağlandıysa c_k değeri sıfırlanmaktadır. Arşivden seçilen yeni birey ise tamamen rastgele seçilmesi yerine c_k değeri 0 olan birey olmaktadır.

4.6. Deneyisel Çalışmalar ve Bulgular

Bu bölümde RESTful web servisler için test senaryosu üretiminde kullanılmak üzere önerilen ABC ve DABC-HS algoritmalarının performansı incelenmiştir. İlk olarak ABC ve DABC-HS algoritmalarının kontrol parametreleri analiz edilmiştir. Ardından bu algoritmaların performansını kapsanan hedefler, bulunan hatalar, üretilen testler ve çalışma zamanı açısından incelemek için literatürdeki problemler olan catwatch, features-service, proxyprint, rest-ncs, rest-news, rest-scs ve scout-api SUT'lar üzerinde test senaryosu üretimi gerçekleştirilmiş ve elde edilen sonuçlar literatürdeki algoritmalarından olan RS, WTS, MIO ve MOSA'nın almış olduğu sonuçlar ile karşılaştırılmıştır.

Bu tez çalışmasında amaç fonksiyonu hesaplanırken dal mesafesi fonksiyonu Tablo 4.1'de verildiği gibi kullanılırken, string ifadeler için de özelleştirilmiş bir versiyonu kullanılmıştır [100]. Tez çalışması Java Sanal Makinası (JVM) ile gerçekleştirilmiştir ve kapsam metriklerine çalışma zamanında erişebilmek amacıyla Java sınıfının bytecode'una doğrudan ea-agent-loader⁴ ve asm⁵ kütüphaneleri eklenmiştir [143].

4.6.1. ABC ve DABC-HS Algoritmalarının Kontrol Parametre Analizi

Gerçek dünya problemleri üzerinde test senaryosu üretimi oldukça maliyetli olduğu için kontrol parametre analizi Gradient, Deceptive ve Plateau isimli Arcuri tarafından önerilmiş [158] yapay problemler üzerinde gerçekleştirilmiştir.

⁴<https://github.com/electronicarts/ea-agent-loader>

⁵<http://asm.ow2.org/>

Algoritmalar toplam z hedef için maksimum değerlendirme sayısı kadar çalışmasına izin verilmiştir. Her bir test, metot çağrı girdisini temsil eden integer ($x \in [0, r]$) bir değer ve metot adını temsil eden bir id olmak üzere iki bileşenden meydana gelmektedir. r , x girdisinin alabileceği maksimum integer sayıyı temsil etmektedir. Her bir hedef k bağımsızdır ve yalnızca $id = k$ testi ile kapsanabilmektedir. Her bir problemin her bir hedefinde rastgele seçilen global optimum ($g \in [0, r]$) bulunmaktadır ve normalizasyon fonksiyonu da $\rho(d) = 1/(1 + d)$ 'dır. Aşağıda belirtildiği gibi üç farklı problem kullanılmıştır.

- **Gradient:** $h_k = \rho(|x - g|)$. En kolay problemlerden biridir. x için global optimum nokta g 'dir.
- **Plateau:** $h_k = \rho(g - x)$ eğer $g \geq x$, aksi takdirde $h_k = \rho(0.1r)$. $g \geq x$ olduğu durumda g optimum noktası için çözüm x olurken aksi takdirde $0.1xr$ olmaktadır.
- **Deceptive:** $h_k = \rho(g - x)$ eğer $g \geq x$, aksi takdirde $h_k = \rho(1 + r - x)$. $g \geq x$ olması durumunda x için global optimum nokta g olurken, aksi takdirde $1 + r - x$ olmaktadır.

Bu çalışmada, ABC ve DABC-HS algoritmalarının kontrol parametrelerini belirlemek için farklı zorlukları benzeten 15 farklı hedef sayısı ($z = \{1, 2, 3, 4, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$) kullanılmıştır. MFE ve r değerleri 1000 olarak seçilmiştir. Her bir çalışma her bir ayar için 10 sefer tekrarlanmıştır. ABC algoritmasının toplam $3 * 3 * 3 = 27$ farklı ayar sayısı varken, DABC-HS algoritmasının $3 * 4 * 3 * 3 * 3 = 324$ farklı ayar sayısı bulunmaktadır. Bu ayarlar Tablo 4.2'de verilmiştir. Böylece en iyi kontrol parametrelerinin belirlenmesi için toplam $(324 + 27) * 10 = 3510$ koşma ve üç problem için toplam 10530 koşma çalıştırılmıştır.

Tablo 4.2. ABC and DABC-HS algoritmaları için araştırılan kontrol parametre setleri.

	ABC	DABC-HS
Popülasyon Büyüklüğü	10, 30, 50	10, 30, 50
Limit	10, 20, 30	5, 10, 15, 20
Çaprazlama Oranı	0.25, 0.50, 0.75	-
<i>ChangeNumber</i>	-	1, 3, 5
<i>ResetNumber</i>	-	3, 5, 7
<i>SizeFactor</i>	-	0.25, 0.5, 0.75 başlangıç noktaları

En iyi kontrol parametre setini belirlemek için sıralama tabanlı göreceli bir değer (Relative Rank-based) Eşitlik 4.4 kullanılarak hesaplanmıştır. *ProjeSayisi* toplam proje sayısını (bu çalışmada 3 adet), *YolSayisi* kaç farklı hedef sayısı bulunduğunu (bu çalışmada 15 adet), $Sonuc_{ayar,i,j}$ ise i. projenin j. hedef sayısının elde ettiği sonucu göstermektedir. ABC ve DABC-HS algoritmaları için en iyi 20 parametre ayarı Tablo 4.3 ve 4.4'te verilmiştir. Tablo 4.3'e göre, *PS*, *limit* ve *CR* kontrol parametrelerinin en iyi sonucu sırasıyla 50, 10 ve 0.50 olmuştur. Tablo 4.4'e göre *PS*, *limit*, *ResetNumber*, ve *ChangeNumber* kontrol parametrelerinin en iyi değerleri sırasıyla 50, 10, 5, 5 olmuştur. Ayrıca en iyi *SizeFactor* değer kümesi ise $\{0.25, 0.75, 0.50\}$ olmuştur. Sonraki deneylerde bu kontrol parametreleri kullanılmıştır.

$$GoreceliSiralama(ayar) = \frac{\sum_{j=1}^{ProjeSayisi} \sum_{i=1}^{YolSayisi} Siralama(Sonuc_{ayar,i,j})}{ProjeSayisi * YolSayisi} \quad (4.4)$$

Tablo 4.3. Her bir parametre için ABC algoritması ile elde edilen sonuçların göreceli sıralama değeri. En iyi ayar seti kalın puntolarla gösterilmiştir. GS: Göreceli Sıralama

	<i>PS</i>	<i>Limit</i>	<i>CR</i>	GS
1	50.00	10.00	0.50	10.90
2	50.00	30.00	0.75	10.91
3	50.00	20.00	0.25	11.48
4	50.00	30.00	0.50	11.48
5	30.00	30.00	0.75	11.76
6	50.00	20.00	0.50	11.93
7	50.00	30.00	0.25	12.14
8	50.00	10.00	0.75	12.34
9	50.00	10.00	0.25	12.39
10	50.00	20.00	0.75	12.68
11	30.00	30.00	0.50	12.91
12	30.00	10.00	0.25	13.22
13	30.00	10.00	0.75	13.46
14	30.00	30.00	0.25	13.91
15	30.00	20.00	0.75	14.11
16	10.00	20.00	0.75	14.36
17	30.00	20.00	0.50	14.57
18	30.00	10.00	0.50	14.87
19	10.00	30.00	0.75	15.19
20	30.00	20.00	0.25	15.48

Tablo 4.4. Her bir parametre için DABC-HS algoritması ile elde edilen sonuçların göreceli sıralama değeri. En iyi ayar seti kalın puntolarla gösterilmiştir. GS: Göreceli Sıralama

	<i>PS</i>	<i>Limit</i>	<i>ResetNumber</i>	<i>ChangeNumber</i>	<i>SizeFactor</i>	<i>GS</i>
1	50.00	10.00	5.00	5.00	[0.25, 0.75, 0.50]	61.72
2	30.00	20.00	5.00	3.00	[0.25, 0.75, 0.50]	61.89
3	30.00	10.00	5.00	5.00	[0.50, 0.25, 0.75]	64.21
4	50.00	10.00	7.00	5.00	[0.25, 0.75, 0.50]	64.79
5	30.00	15.00	5.00	3.00	[0.25, 0.75, 0.50]	65.03
6	50.00	5.00	7.00	5.00	[0.25, 0.75, 0.50]	66.53
7	30.00	20.00	5.00	5.00	[0.50, 0.25, 0.75]	67.24
8	30.00	5.00	5.00	5.00	[0.50, 0.25, 0.75]	67.58
9	50.00	20.00	5.00	5.00	[0.25, 0.75, 0.50]	67.80
10	30.00	15.00	7.00	5.00	[0.50, 0.25, 0.75]	68.71
11	50.00	5.00	5.00	5.00	[0.25, 0.75, 0.50]	70.17
12	30.00	10.00	3.00	3.00	[0.50, 0.25, 0.75]	70.23
13	50.00	20.00	7.00	5.00	[0.25, 0.75, 0.50]	70.93
14	30.00	10.00	7.00	5.00	[0.25, 0.75, 0.50]	71.33
15	30.00	10.00	5.00	3.00	[0.25, 0.75, 0.50]	74.76
16	50.00	15.00	5.00	5.00	[0.25, 0.75, 0.50]	74.81
17	30.00	5.00	7.00	5.00	[0.25, 0.75, 0.50]	74.92
18	50.00	15.00	7.00	5.00	[0.25, 0.75, 0.50]	75.61
19	30.00	20.00	3.00	3.00	[0.50, 0.25, 0.75]	77.02
20	30.00	20.00	7.00	5.00	[0.25, 0.75, 0.50]	78.16

4.6.2. Literatürdeki Algoritmalarla Karşılaştırma

DABC-HS ve temel ABC algoritmalarının performansını incelemek için gerçek dünya problemleri üzerinde kapsanan hedef sayısı, hata sayısı, test sayısı ve çalışma zamanı sonuçları alınmış, MIO [158], MOSA [179], WTS [54] ve RS algoritmalarının elde ettiği sonuçlarla kıyaslanmıştır.

Deneylerde, bütün algoritmalar [158] çalışmasında kullanılan 7 farklı açık kaynaklı SUT üzerinde çalıştırılmıştır. Kullanılan SUT'lara Github⁶ üzerinden erişilebilmektedir ve SUT'lar hakkında genel bilgi Tablo 4.5'te verilmiştir. Bu SUT'lardan catwatch, features-service, proxyprint ve scout-api projeleri Github üzerinden seçilen açık kaynaklı piyasada kullanılan projelerdir. Rest-news, rest-ncs ve rest-scs ise Arcuri [158] tarafından SpringBoot ile oluşturulmuş yapay projelerdir.

⁶<https://github.com/EMResearch/EMB>

Tablo 4.5. Önerilen yöntemlerin analizinde kullanılan SUT'ların özellikleri.

Adı	Sınıf Sayısı	Satır Sayısı	Uç Nokta Sayısı	Veritabanı
catwatch	69	5442	23	var
features-service	23	1247	18	var
proxyprint	68	7534	115	var
rest-ncs	9	602	6	yok
rest-news	10	715	14	var
rest-scs	13	859	11	yok
scout-api	75	7479	49	var
Toplam	267	23878	236	5/7 var

Her bir deneyde maksimum 100000 HTTP çağrısı durdurma kriteri kullanılmıştır. Her bir deney 30 farklı seed sayısı ile çalışmış olup toplam 595 günlük CPU zamanına ihtiyaç duymaktadır.

Çalışmada kullanılan MIO, MOSA ve WTS algoritmalarının kontrol parametreleri [158]'de olduğu gibi kullanılmıştır. MOSA ve WTS için popülasyon sayısı 50'dir. MIO için F , P_r ve n kontrol parametreleri sırasıyla 0.5, 0.5 ve 10 iken, MOSA için turnuva sayısı 10'dur. WTS için maksimum test senaryo sayısı 50'dir. Her bir mutasyon operatörü (ekleme, silme, değiştirme) için oran ise $1/3$ 'tür olurken, WTS için CR 0.7'dir. Önerilen algoritmada, MIO'da olduğu gibi değişken sayıda mutasyon için kullanılan, maksimum, minimum, *threshold* ve n parametreleri sırasıyla 10, 1, 0.5 ve 10'dur (Eşitlik 4.3).

Deneylerden elde edilen ortalama kapsama miktarı Tablo 4.6'da verilmiştir. Daha iyi sütununda, Mann-Whitney-Wilcoxon U-testi sonucu $p < 0.05$ olan algoritmalar gösterilmektedir. Algoritma isimlerinin yanında parantez içerisinde ise Vargha-Delaney etki büyüklüğü ($\hat{A}_{12}$) verilmiştir. Tablo 4.7'de ise Friedman test sonucuna göre göreceli sıralama değerleri sunulmaktadır. Bir (1) değeri erişilen en yüksek kapsama sayısını temsil etmektedir. Friedman testi sonucunda elde edilen X^2 ve p-değeri verilmiştir.

Tablo 4.6 ve 4.7 incelendiğinde, DABC-HS algoritmasının 7 SUT içerisinde 4'ünde en iyi performansı gösterdiği görülmektedir. DABC-HS bir problemde ikinci olurken, kalan iki problemde de üçüncü olmuştur. RS algoritmasının performansı, kapsanan hedef sayısı dikkate alındığında, bütün problemlerde diğer algoritmaların gerisinde kalmıştır.

Tablo 4.6. Deneylerde elde edilen ortalama kapsama miktarı. Daha iyi sütununda, Mann-Whitney-Wilcoxon U-testi sonucu $p < 0.05$ olan algoritmalar verilmiştir. Algoritma isimlerinin yanında parantez içerisinde ise Vargha-Delaney etki büyüklüğü ($\hat{A}_{12}$) verilmiştir.

SUT	Algoritma	Kapsama	Daha İyi
<i>catwatch</i>	ABC	1065.5	RS (0.05)
	MIO	1109.7	ABC (0.11) RS (0.00)
	MOSA	1180.6	ABC (0.00) DABC-HS (0.17) MIO (0.03) RS (0.00) WTS (0.02)
	RS	963.0	N/A
	WTS	1118.4	ABC (0.01) RS (0.00)
	DABC-HS	1137.7	ABC (0.00) MIO (0.11) RS (0.00) WTS (0.08)
<i>features-service</i>	ABC	504.4	RS (0.00)
	MIO	560.0	ABC (0.17) RS (0.00)
	MOSA	562.9	ABC (0.18) MIO (0.29) RS (0.00)
	RS	237.1	N/A
	WTS	561.7	ABC (0.20) MIO (0.33) MOSA (0.25) RS (0.00)
	DABC-HS	620.5	ABC (0.11) MIO (0.15) MOSA (0.10) RS (0.00) WTS (0.14)
<i>proxypoint</i>	ABC	1399.4	N/A
	MIO	1401.8	N/A
	MOSA	1405.0	ABC (0.30) RS (0.30)
	RS	1393.4	N/A
	WTS	1463.0	ABC (0.18) MOSA (0.22) RS (0.18)
	DABC-HS	1493.0	ABC (0.24) MIO (0.24) MOSA (0.20) RS (0.24) WTS (0.28)
<i>rest-ncs</i>	ABC	476.9	RS (0.00)
	MIO	528.6	ABC (0.00) RS (0.00) WTS (0.09)
	MOSA	527.5	ABC (0.00) RS (0.00) WTS (0.10)
	RS	369.5	N/A
	WTS	519.2	ABC (0.00) RS (0.00)
	DABC-HS	529.8	ABC (0.00) MOSA (0.33) RS (0.00) WTS (0.06)
<i>rest-news</i>	ABC	289.9	RS (0.07)
	MIO	289.9	ABC (0.33) RS (0.04) WTS (0.19)
	MOSA	292.5	ABC (0.25) RS (0.02) WTS (0.11)
	RS	260.2	N/A
	WTS	285.7	RS (0.09)
	DABC-HS	300.1	ABC (0.19) MIO (0.29) MOSA (0.34) RS (0.01) WTS (0.06)
<i>rest-scs</i>	ABC	517.2	RS (0.27)
	MIO	590.8	ABC (0.03) RS (0.00) WTS (0.01)
	MOSA	762.7	ABC (0.00) DABC-HS (0.00) MIO (0.00) RS (0.00) WTS (0.00)
	RS	509.6	N/A
	WTS	533.0	ABC (0.04) RS (0.00)
	DABC-HS	577.4	ABC (0.03) RS (0.00) WTS (0.11)
<i>scout-api</i>	ABC	1943.7	DABC-HS (0.00) MIO (0.00) MOSA (0.03) RS (0.00) WTS (0.03)
	MIO	1753.8	RS (0.00)
	MOSA	1720.2	RS (0.00)
	RS	1419.1	N/A
	WTS	1910.8	DABC-HS (0.06) MIO (0.02) MOSA (0.03) RS (0.00)
	DABC-HS	1830.3	MIO (0.26) MOSA (0.19) RS (0.00)

Temel ABC algoritması scout-api ve rest-news problemleri hariç diğer problemlerde en kötü ikinci algoritma olurken, scout-api probleminde birinci, rest-news probleminde ise üçüncü sıradadır. MOSA, catwatch ve rest-scs problemlerinde ilk sırada bulunurken, features-service ve rest-news problemlerinde ikinci sırada, proxyprint ve rest-ncs problemlerinde üçüncü sırada ve scout-api probleminde beşinci sıradadır. MIO, rest-ncs ve rest-scs problemlerinde ikinci sırada, rest-news probleminde üçüncü ve diğer problemlerde dördüncü sırada bulunmaktadır. Son olarak WTS ise proxyprint ve scout-api problemlerinde ikinci sırada bulunurken, catwatch ve features-service problemlerinde üçüncü ve diğer problemlerde dördüncü sıradadır.

Tablo 4.7’de görüldüğü gibi en iyi sıralama değerine 12 ile DABC-HS sahip olurken, ardından 17 ile MOSA gelmektedir. WTS ve MIO ise sırasıyla 22 ve 23 sıralama değerine sahiptir. Bu sonuçlara göre DABC-HS algoritmasının kapsanan hedef sayısına göre daha gülbüz sonuçlar ürettiği görülmektedir ve istatistiksel test sonucunda elde edilen p-değerine göre (Tablo 4.7) de bu sonuç desteklenmektedir.

Tablo 4.7. Algoritmaların sıralama tabanlı ortalama performansları.

SUT	Algoritmalar					
	ABC	MIO	MOSA	RANDOM	WTS	DABC-HS
<i>catwatch</i>	5	4	1	6	3	2
<i>features-service</i>	5	4	2	6	3	1
<i>proxyprint</i>	5	4	3	6	2	1
<i>rest-ncs</i>	5	2	3	6	4	1
<i>rest-news</i>	3	3	2	5	4	1
<i>rest-scs</i>	5	2	1	6	4	3
<i>scout-api</i>	1	4	5	6	2	3
TOPLAM SIRALAMA	29	23	17	41	22	12
$X^2 = 22.418, p - \text{degeri} = 0.0004359$						

Algoritmaların ortalama 5xx hata sayısı, üretilen test sayısı ve ortanca zaman analizleri sırasıyla Tablo 4.8-4.10’da verilmiştir. DABC-HS pivot algoritma olarak seçilirken, DABC-HS ile diğer algoritmalar arasındaki Vargha-Delaney etki büyüklüğü $\hat{A}_{12}$ alt satırda, parantez içerisinde verilmiştir. Mann-Whitney-Wilcoxon U-testi sonucu $\alpha = 0.05$ güven aralığında anlamlı ise kalın puntolarla yazılmıştır.

Tablo 4.8’deki ortalama hata sayısı incelendiğinde, rest-ncs, rest-news, rest-scs ve

scout-api problemlerinde bütün algoritmalar aynı sayıda hata bulmuştur ve sonuçlarda anlamlı bir farklılık bulunmamaktadır. Catwatch probleminde, RS hatalardan birini bulamamıştır. Features-service probleminde DABC-HS ve WTS algoritmaları diğer algoritmalarından daha fazla hata bulurken, DABC-HS, WTS hariç diğer algoritmalarından istatistiksel olarak daha başarılıdır. Proxyprint probleminde, WTS en fazla hatayı bulan algoritma olurken DABC-HS de oldukça yakın sonuç elde etmiştir ve aralarında istatistiksel olarak farklılık bulunmamaktadır. Genel sıralama incelendiğinde, DABC-HS, WTS hariç diğer algoritmalarından daha iyi performans göstermiştir. WTS en fazla hata bulan algoritma olurken, MIO üçüncü sıradadır. ABC ve MOSA aynı sıralama değerine sahipken, RS diğer algoritmaların gerisinde, son sırada kalmıştır.

Tablo 4.8. 5xx hata sayılarının ortalama değerleri. DABC-HS pivot algoritmadır ve DABC-HS ile diğer algoritmalar arasındaki Vargha-Delaney etki büyüklüğü $\hat{A}_{12}$ alt satırda, parantez içerisinde verilmiştir. Mann-Whitney-Wilcoxon U-testi sonucu $\alpha = 0.05$ güven aralığında anlamlı ise kalın puntolarla yazılmıştır.

SUT	DABC-HS	ABC	MIO	MOSA	RS	WTS
<i>catwatch</i>	6.0	6.0 (0.5)	6.0 (0.5)	6.0 (0.5)	5.0 (0.0)	6.0 (0.5)
<i>features-service</i>	14.0	13.7 (0.3)	13.3 (0.1)	13.3 (0.1)	13.0 (0.0)	14.0 (0.5)
<i>proxyprint</i>	29.9	28.4 (0.0)	29.1 (0.2)	28.6 (0.1)	28.0 (0.0)	30.0 (0.6)
<i>rest-ncs</i>	0.0	0.0 (0.5)	0.0 (0.5)	0.0 (0.5)	0.0 (0.5)	0.0 (0.5)
<i>rest-news</i>	1.0	1.0 (0.5)	1.0 (0.5)	1.0 (0.5)	1.0 (0.5)	1.0 (0.5)
<i>rest-scs</i>	0.0	0.0 (0.5)	0.0 (0.5)	0.0 (0.5)	0.0 (0.5)	0.0 (0.5)
<i>scout-api</i>	33.0	33.0 (0.5)	33.0 (0.5)	33.0 (0.5)	33.0 (0.5)	33.0 (0.5)
TOPLAM SIRALAMA	8	12	11	12	16	7

$X^2 = 11.867, p - \text{degeri} = 0.03665$

Tablo 4.9’da verilen üretilen test sayısı ortalamaları incelendiğinde, catwatch probleminde en fazla test üreten algoritma DABC-HS olsa da kapsanan hedef sayısı açısından MOSA’nın gerisinde kalmıştır. MOSA daha fazla hedefi daha az test ile kapsamayı başarmıştır. RS en az test üreten ve en az hedef kapsayan algoritma olmuştur. Features-service probleminde en fazla test üreten ve en fazla kapsama miktarına erişen

algoritma DABC-HS'dir. Bu problemde RS en az test sayısına ve kapsama miktarına sahip algoritma olmuştur. Bu problemdeki en önemli bulgulardan biri de WTS, DABC-HS'den sonra en fazla test senaryosu üreten algoritma olurken, kapsama olarak RS'den sonra en kötü algoritma olmuştur. Proxyprint probleminde elde edilen sonuçlara göre, DABC-HS, MIO'dan sonra en az test senaryosu üreten algoritma olurken, en fazla kapsama miktarına erişen algoritma olmayı da başarmıştır. MIO ise dördüncü sırada bulunmaktadır. En fazla test senaryosuna sahip algoritma olan MOSA ise kapsama miktarı açısından üçüncü sıradadır. rest-ncs probleminde, en fazla test senaryosu üreten algoritmalar DABC-HS ve MIO'dur ve kapsama miktarı açısından da ilk iki sırayı paylaşmıştır. En az test senaryosunu RS üretirken aynı zamanda en kötü performansa sahip olan algoritma olmuştur. Bu problemde üretilen test sayısı ve kapsama miktarı arasında doğru orantı bulunmaktadır. Rest-news probleminde, WTS en fazla test senaryosu üreten algoritma olurken RS'ten sonra en kötü performans gösteren algoritma olmuştur. DABC-HS ise üretilen test senaryosu açısından ikinci sırada bulunurken en fazla hedef kapsama başarısını gösteren algoritmadır. MIO ve MOSA algoritmalarının ortalama test senaryo sayısı ve kapsama miktarları birbirine benzerdir. Rest-scs problemi incelendiğinde, en fazla test senaryosu MOSA'ya aittir ve hedef kapsama açısından da ilk sıradadır. Test senaryo sayısı açısından MOSA'nın ardından ise MIO, DABC-HS, WTS, ABC ve RS gelmektedir. Kapsama miktarları test senaryo sayısı ile doğru orantılıdır. Son olarak scout-api problemi incelendiğinde, en fazla test sayısı ABC algoritmasına aittir ve en fazla hedef kapsayan algoritma olmuştur. Hemen ardından test senaryosu ve kapsama miktarı açısından WTS gelmektedir. DABC-HS bu iki algoritmanın ardından gelirken, MIO en az test senaryosu üreten algoritma olmuştur ancak MOSA'dan daha fazla hedef kapsamayı başarmıştır. MOSA ise RS'den daha az test senaryosu üretmiş olsa da RS'yi geçmeyi başarmıştır. Toplam sıralama değerleri incelendiğinde, RS en az test senaryosu üreten algoritma olmuştur. Ardından ABC, MIO, WTS, MOSA ve DABC-HS gelmektedir. Genel olarak, daha fazla hedef kapsayabilmek için daha fazla test senaryosu üretmek gerektiği söylenebilmektedir.

Tablo 4.9. Üretilen test sayılarının ortalama değerleri. DABC-HS pivot algoritmadır ve DABC-HS ile diğer algoritmalar arasındaki Vargha-Delaney etki büyüklüğü $\hat{A}_{12}$ alt satırda, parantez içerisinde verilmiştir. Mann-Whitney-Wilcoxon U-testi sonucu $\alpha = 0.05$ güven aralığında anlamlı ise kalın puntolarla yazılmıştır.

SUT	DABC-HS	ABC	MIO	MOSA	RS	WTS
<i>catwatch</i>	51.8	40.1 (0.0)	43.6 (0.0)	46.3 (0.1)	39.2 (0.0)	43.7 (0.0)
<i>features-service</i>	67.2	37.9 (0.0)	35.3 (0.0)	50.8 (0.0)	26.0 (0.0)	65.6 (0.4)
<i>proxyprint</i>	207.1	228.9 (1.0)	202.0 (0.2)	242.2 (1.0)	234.9 (1.0)	233.1 (1.0)
<i>rest-ncs</i>	57.0	40.5 (0.0)	56.6 (0.4)	51.2 (0.1)	35.9 (0.0)	49.9 (0.0)
<i>rest-news</i>	25.6	22.3 (0.1)	23.7 (0.2)	23.7 (0.2)	22.7 (0.1)	25.9 (0.6)
<i>rest-scs</i>	70.6	37.1 (0.0)	73.8 (0.6)	123.7 (1.0)	33.8 (0.0)	53.3 (0.1)
<i>scout-api</i>	184.1	199.2 (1.0)	169.7 (0.0)	177.4 (0.2)	177.8 (0.2)	193.9 (0.9)
TOPLAM SIRALAMA	32	19	20	30	14	29
$X^2 = 12.09, p - \text{degeri} = 0.03357$						

Tablo 4.10’da verilen zaman kıyaslamalarına göre, RS bütün algoritmalar içerisinde en hızlı çalışan algoritma olmuştur. MOSA ise rest-ncs ve rest-news problemleri hariç en yavaş çalışan algoritmadır. Bu iki problemde ise MIO, MOSA’dan daha yavaş çalışmıştır. Catwatch probleminde, ikinci en hızlı algoritma MIO olurken ardından DABC-HS, WTS ve MOSA gelmektedir. Features-service probleminde ise ikinci en hızlı algoritma DABC-HS olurken ardından WTS, ABC ve MIO gelmiştir. Proxyprint probleminde, WTS ve ABC algoritmaları iki ve üçüncü sırada bulunurken bu algoritmaların ardından DABC-HS ve MIO gelmiştir. Res-ncs probleminde, RS algoritmasının ardında sırasıyla DABC-HS, WTS, ABC ve MOSA algoritmaları bulunmaktadır. Rest-news probleminde ikinci en hızlı algoritma WTS’dir ve ardından ABC, MOSA ve DABC-HS gelmektedir. Rest-scs probleminde MIO ikinci en hızlı algoritma olurken DABC-HS de üçüncü sırada bulunmaktadır. WTS ve ABC algoritmaları ise dört ve beşinci sıraları paylaşmaktadır. Son olarak scout-api probleminde DABC-HS algoritması ikinci olurken, ardından sırasıyla WTS, ABC ve MIO gelmektedir. Toplam sıralama değeri incelendiğinde en hızlı algoritma RS olurken daha sonra ikinci sırayı DABC-HS ve WTS algoritmaları

paylaşmıştır. ABC üçüncü sırada bulunurken son iki sırada sırasıyla MIO ve MOSA bulunmaktadır.

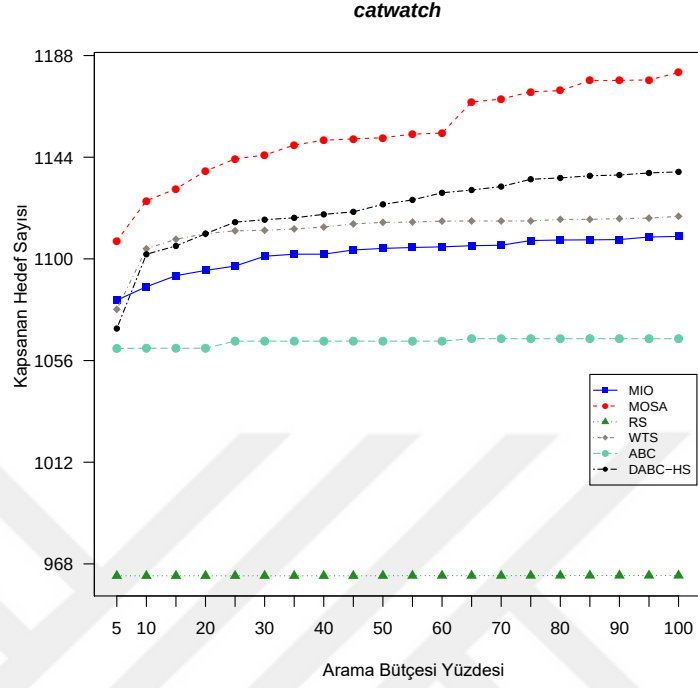
Tablo 4.10. Çalışma zamanının ortanca değerleri. DABC-HS pivot algoritmadır ve DABC-HS ile diğer algoritmalar arasındaki Vargha-Delaney etki büyüklüğü $\hat{A}_{12}$ alt satırda, parantez içerisinde verilmiştir. Mann-Whitney-Wilcoxon U-testi sonucu $\alpha = 0.05$ güven aralığında anlamlı ise kalın puntolarla yazılmıştır.

SUT	DABC-HS	ABC	MIO	MOSA	RS	WTS
<i>catwatch</i>	8081.5	10044.0 (1.0)	7953.5 (0.4)	14092.5 (1.0)	7230.0 (0.0)	9147.0 (1.0)
<i>features-service</i>	8659.0	9770.0 (1.0)	9785.0 (1.0)	10989.0 (1.0)	7670.5 (0.0)	9383.0 (1.0)
<i>proxyprint</i>	21340.0	18682.5 (0.0)	27079.5 (1.0)	38718.5 (0.8)	14561.5 (0.0)	14903.0 (0.0)
<i>rest-ncs</i>	9616.5	10376.5 (1.0)	12937.0 (1.0)	12557.0 (1.0)	8119.0 (0.0)	9955.0 (0.8)
<i>rest-news</i>	9403.0	7954.5 (0.0)	15589.0 (1.0)	9290.5 (0.5)	6769.0 (0.0)	7612.0 (0.0)
<i>rest-scs</i>	7425.5	10037.5 (1.0)	7292.5 (0.5)	12296.5 (1.0)	6688.0 (0.2)	7890.0 (0.7)
<i>scout-api</i>	8488.0	9176.5 (0.9)	9919.5 (1.0)	14389.0 (1.0)	7844.5 (0.1)	8497.5 (0.5)
TOPLAM SIRALAMA	21	28	31	39	7	21
$X^2 = 24.306, p - \text{degeri} = 0.0001896$						

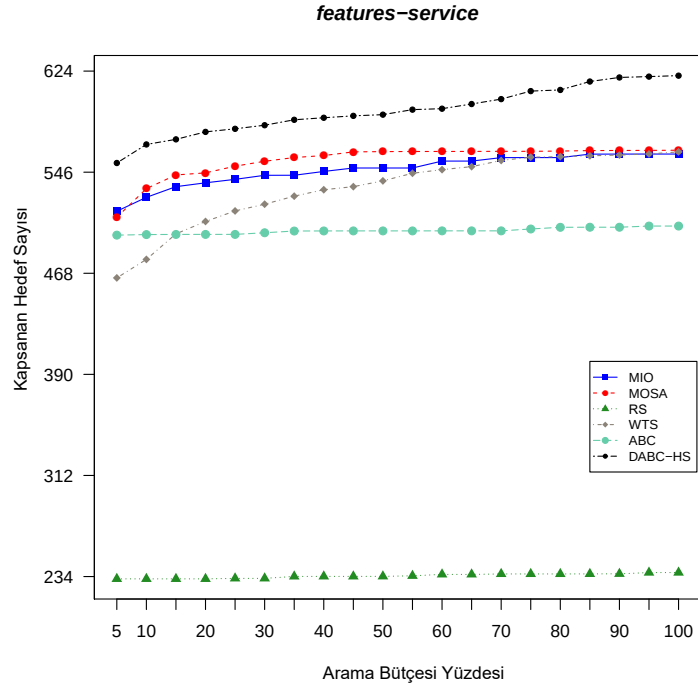
Algoritmaların kapsadığı hedef sayısı açısından kapsamlı performans analizini gerçekleştirmek amacıyla, her bir problemde elde edilen arama sürecinin çeşitli yüzdelinde kapsanan hedef sayıları Şekil 4.13-4.19’da verilmiştir. Grafikler genel olarak incelendiğinde, RS’in hiçbir problemde kapsadığı hedef sayısını iyileştiremediği görülmektedir. Şekil 4.13’te verilen catwatch problemi incelendiğinde, MOSA ve DABC-HS hariç hiçbir algoritma %25’lik HTTP çağrı sayısı diliminden sonra kapsadığı hedef sayısını iyileştirememiştir. MOSA’nın onuncu yüzdilik dilimde elde ettiği sonuç, DABC-HS hariç diğer algoritmaların en sonda elde ettiği sonuçtan daha iyidir. DABC-HS ise MOSA kadar performansını iyileştiremese de diğer algoritmalarından daha iyi performans göstermiş ve süreç boyunca da iyileştirmeye devam etmiştir. Features-service problemi (Şekil 4.14) incelendiğinde ilk yüzde beşlik dilim sonunda DABC-HS algoritması diğer algoritmaların arama boyunca elde edebildiği sonuçtan daha iyi kapsama miktarına erişmiştir ve arama sürecinde de yeni hedefler kapsamaya

devam etmiştir. MIO, MOSA ve WTS algoritmaları ise yüzde altmışlık dilimden sonra benzer kapsama sayısına erişmiştir. ABC ve RS algoritmaları ise arama başından itibaren kapsadığı hedef sayısını çok fazla artırmayı başaramamıştır. Şekil 4.15'te verilen proxyprint sonuçlarına göre, MOSA en kötü başlangıcı yapan algoritma olsa da kırk beşinci yüzdelerik dilimle birlikte MIO, ABC, WTS ve RS algoritmalarını geçmeyi başarmıştır. ABC ve RS ise WTS hariç en iyi başlangıcı yapan algoritmalar olsa da devam eden süreçte çok fazla gelişim gösterememiş ve diğer algoritmaların gerisinde kalmışlardır. MIO ilk yüzde otuz beşlik dilimde gelişim gösterse de aramanın devamında kapsadığı hedef sayısını artıramamıştır. En iyi başlangıcı yapan algoritma ise WTS olsa da DABC-HS, WTS'i ilk yüzde on beşlik dilimden sonra geçmiştir ve süreç boyunca da kapsadığı hedef sayısını artırmıştır. Şekil 4.16'da verilen Rest-ncs probleminin analizlerine göre MIO ve DABC-HS algoritmaları benzer başlangıç yapmıştır ve arama süreci boyunca da performansları benzemeye devam etmektedir. MOSA, catwatch probleminde olduğu gibi en iyi başlangıcı yapan algoritmadır. WTS başlangıçta ABC algoritmasının gerisinde kalsa da yüzde onluk HTTP çağrısından sonra ABC algoritmasını geride bırakmış fakat DABC-HS, MOSA ve MIO'nun gerisinde kalmıştır. Rest-news için elde edilen sonuçlara (Şekil 4.17) göre, DABC-HS ilk yüzde ellilik dilimde oldukça önemli bir iyileşme göstermiş ve diğer algoritmaları geride bırakmıştır. ABC en iyi başlangıcı yapan algoritma olsa da sonraki arama sürecinde çok fazla iyileşme gösterememiştir. MOSA oldukça kötü bir başlangıç yapsa da yüzde kırk beşinci yüzdelerik dilimden sonra DABC-HS hariç diğer algoritmaları geçmiştir. WTS ve MIO ise başlangıçta benzer performans göstermiştir ancak daha sonra MIO, WTS algoritmasını geçmeyi başarmıştır. Şekil 4.18'de verilen rest-scs problemine göre MOSA'nın diğer algoritmalara üstünlüğü görülmektedir. Diğer algoritmalar başlangıçta benzer kapsama miktarlarına sahip olsa da, arama sonunda MIO ikinci sıraya yükselmiştir. DABC-HS, MOSA ve MIO süreç boyunca kapsadığı hedef sayısını iyileştirmeye devam etmiştir. ABC, WTS ve RS yüzde ellilik dilimden sonra kapsadığı hedef sayısını iyileştirememiştir. Scout-api probleminde (Şekil 4.19) diğer problemlerdeki sonuçlardan biraz farklı sonuç elde edilmiştir. ABC ve WTS algoritmaları diğer algoritmalarından daha iyi performans sergilemiştir. DABC-HS başlangıçta iyi bir gelişme gösterse de bu iki algoritmaya yetişememiştir. MIO yüzde otuzluk dilimden sonra kapsadığı hedef sayısını iyileştirememiştir. MOSA ise RS'den sonra en az hedef kapsayan algoritmadır ve arama

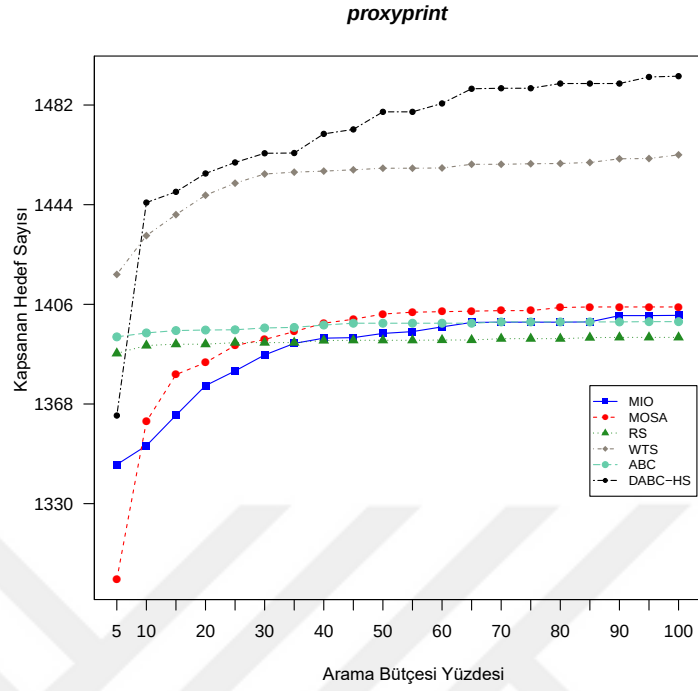
boyunca kapsadığı hedef sayısını artırsa da diğer algoritmaların gerisinde kalmıştır.



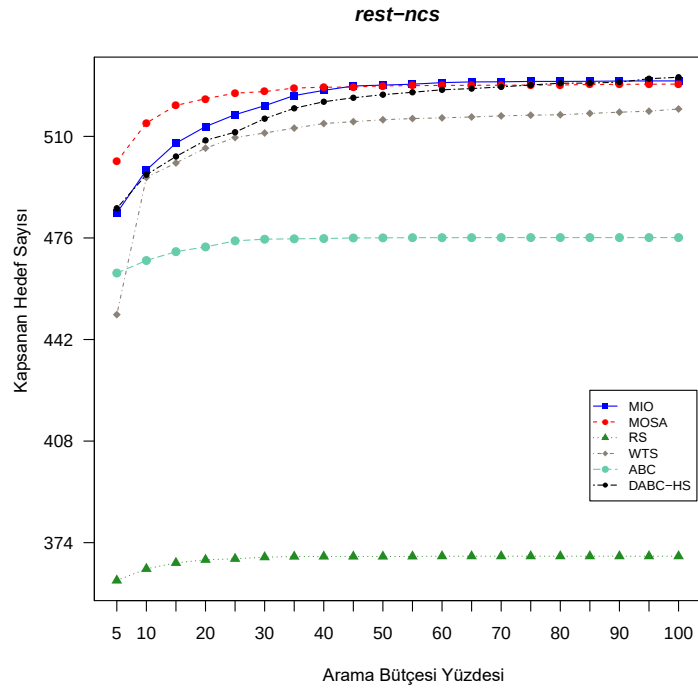
Şekil 4.13. Farklı arama algoritmalarının catwatch problemi üzerinde elde etmiş oldukları sonuçların grafiksel gösterimi.



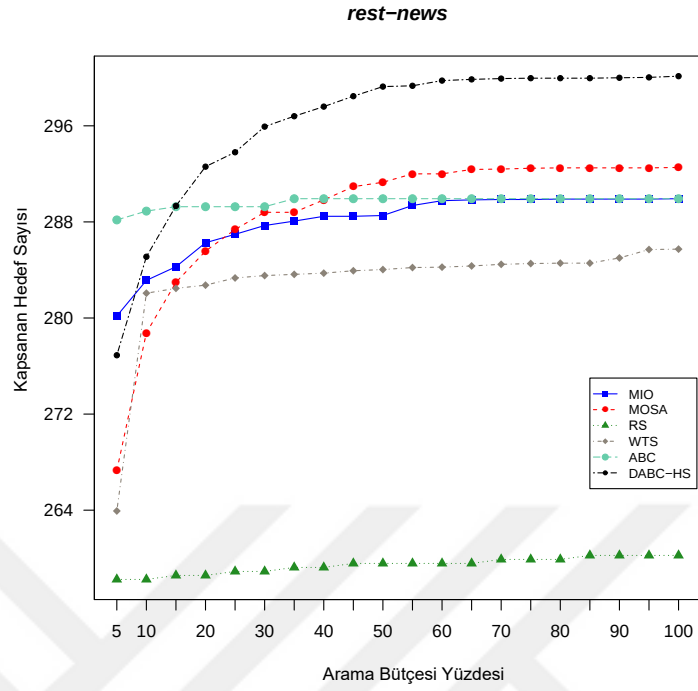
Şekil 4.14. Farklı arama algoritmalarının Features-service problemi üzerinde elde etmiş oldukları sonuçların grafiksel gösterimi.



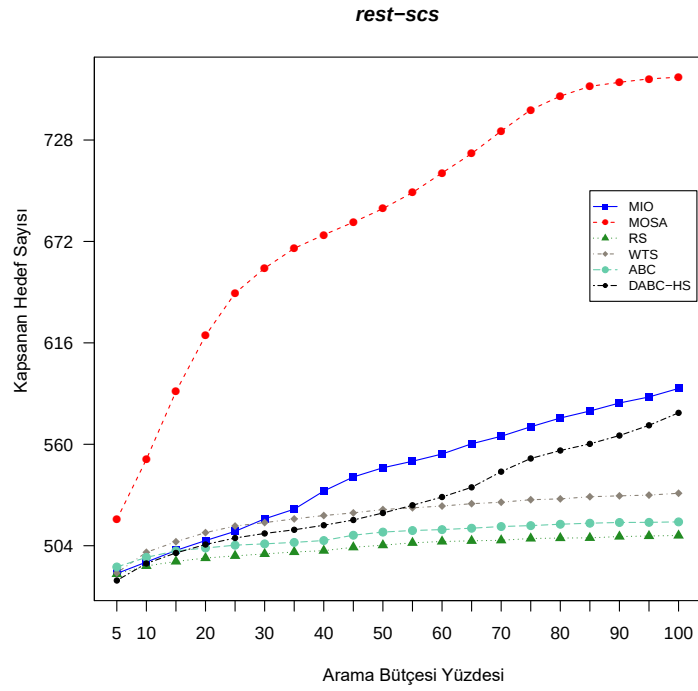
Şekil 4.15. Farklı arama algoritmalarının Proxyprint problemi üzerinde elde etmiş oldukları sonuçların grafiksel gösterimi.



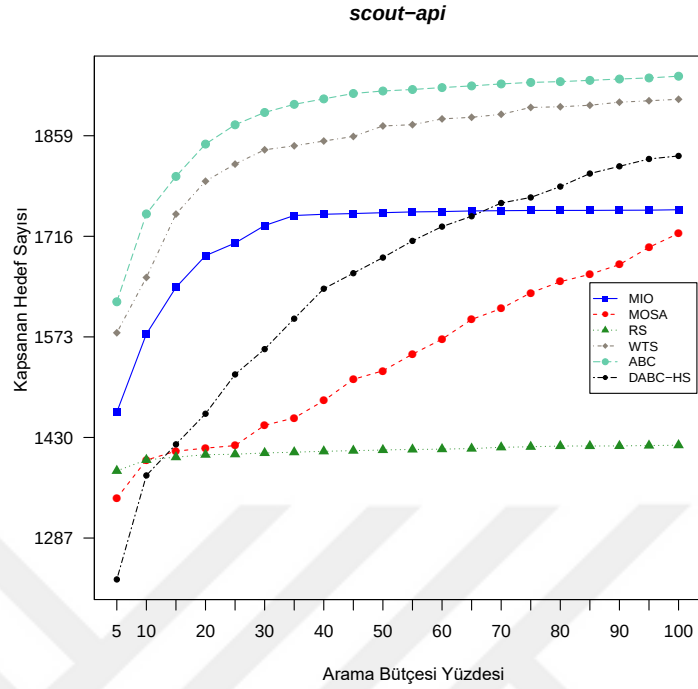
Şekil 4.16. Farklı arama algoritmalarının Rest-ncs problemi üzerinde elde etmiş oldukları sonuçların grafiksel gösterimi.



Şekil 4.17. Farklı arama algoritmalarının Rest-news problemi üzerinde elde etmiş oldukları sonuçların grafiksel gösterimi.



Şekil 4.18. Farklı arama algoritmalarının Rest-scs problemi üzerinde elde etmiş oldukları sonuçların grafiksel gösterimi.



Şekil 4.19. Farklı arama algoritmalarının Scout-api problemi üzerinde elde etmiş oldukları sonuçların grafiksel gösterimi.

4.6.3. Değerlendirme

Bu bölümde ilk olarak, ABC algoritması RESTful web servisler için test senaryosu üretimine uygun hale getirilmiştir. Ardından ABC algoritmasındaki eksikliklere göre çeşitli mekanizmaların önerildiği DABC-HS algoritması geliştirilmiştir. Tek hedefin optimize edildiği ABC algoritmasının aksine önerilen yöntem; kapsama miktarını, hata sayısını ve test sayısını dikkate alan ve çok amaçlı bir optimizasyon algoritmasıdır. Ayrıca, hiper-kâşif arı fazı eklemesiyle birlikte ABC algoritmasının keşif yeteneği de artırılmıştır. Dinamik görevli/gözcü arı sayısı da keşif ve faydalanma arasındaki dengeyi kurmada yardımcı olmuştur.

Deneylerin ilk kısmında ABC ve DABC-HS algoritmalarının kontrol parametreleri ayarlanmıştır. Kontrol parametrelerinin ayarlanmasının ardından MIO, MOSA, WTS ve RANDOM algoritmalarıyla 23 binden fazla satır koda sahip yedi farklı web servis üzerinde kıyaslanmış, istatistiksel analizler gerçekleştirilmiştir. Elde edilen sonuçlar göstermiştir ki DABC-HS algoritması en fazla hedef kapsayan algoritma olurken, yedi problemin dördünde diğer algoritmaları geride bırakmıştır. Diğer problemlerde ise diğer

algoritmalarla oldukça yakın performans sergilemiştir. Ayrıca önerilen yöntem SUT'lar üzerinde 84 hata bulmuştur. Çalışma zamanına göre incelendiğinde ise RS kadar hızlı olmasa da WTS ile benzer hıza sahiptir. Önerilen algoritma çok amaçlı algoritmalar olan MIO'dan ve MOSA'dan daha hızlıdır. Bu da önerilen algoritmanın aynı zaman diliminde daha fazla iterasyon çalıştığını göstermektedir. Test senaryosu üretiminin oldukça hesaplama zamanı yüksek bir eylem olduğu düşünüldüğünde algoritmaların hızlı çalışıyor olmasının önemi ortaya çıkmaktadır.

Önerilen algoritma, bütün problemlerde en iyi sonuca erişemese de literatürde önerilen diğer algoritmalarla daha başarılı olmuştur. Bu yöntem diğer kombinasyonel problemleri çözmek için de önemli bir potansiyel taşımaktadır. Ayrıca literatürde RESTful web servisler için test senaryosu üretimi çalışmaları oldukça kısıtlıdır ancak endüstride bu alanda yapılacak olan çalışmalara ihtiyaç her geçen gün daha da artmaktadır. Dolayısıyla bu çalışmanın literatüre katkısı oldukça önemlidir.

Gerçekleştirilen bu çalışmanın birkaç eksik kaldığı nokta bulunmaktadır. Bunlardan biri kontrol parametrelerinin yapay problemler üzerinde ayarlanmasıdır. Gerçek dünya problemlerinde algoritmaların performansı etkilenebildiği için kontrol parametrelerinin gerçek dünya problemleri üzerinde ayarlanmasının daha iyi sonuç elde etmeye faydasının olabildiği düşünülmektedir. Sistem kaynaklarının yeterli olduğu durumda gerçek dünya problemleri ile kontrol parametreleri ayarlanmalıdır ancak hesaplama maliyeti yüksekliği sebebiyle mevcut koşullardaki hesaplama gücü ile gerçekleştirilmesi mümkün değildir. Ayrıca algoritmaların performansı rastgelelikten etkilenmektedir. Bu gerekçe nedeniyle farklı seed sayıları ile 30 farklı deney gerçekleştirip uygun istatistiksel analizlerle sonuçlar sunulmuştur.

Gelecek çalışmalarda ise problem karakteristiğine göre uygun mekanizmalar ile algoritmaların performansı iyileştirilebilir. GraphQL sorgu dili gibi popüler yaklaşımların da desteklenmesi gelecek çalışmalarda planlanmaktadır.

5. BÖLÜM

TARTIŞMA, SONUÇ ve ÖNERİLER

5.1. Tartışma ve Sonuçlar

Bu tez çalışmasında, SBST problemlerinden biri olan test senaryosu üretimi için ABC tabanlı algoritmalar geliştirilmiştir. ABC algoritmasının SBST problemlerinin bir türü olan test verisi üretimi problemindeki başarısı [103] sebebiyle test senaryosu üretimi üzerindeki performansı da incelenmiş ve eksiklikleri düşünülerek gerekli değişiklikler gerçekleştirilmiştir. Çalışma, nesne yönelimli programlama için test senaryosu üretimi ve RESTful web servisler için test senaryosu üretimi olmak üzere iki farklı alan üzerinde yürütülmüştür.

Nesne yönelimli programlamanın; kalıtım, kapsülleme, çok-biçimlilik, soyutlama, bilgi gizleme gibi özellikleri sebebiyle test süreçleri zorlaşsa da uygulamaların test edilmesi ve hataların olabildiği kadar hızlı tespit edilmesi oldukça önemlidir. Bu çalışmada, nesne yönelimli programlama için test senaryosu üretimine uygun iki farklı ABC algoritması geliştirilmiştir. ABC algoritması çaprazlama ve mutasyon operatörleri kullanılarak bir kombinasyonel problem türü olan test senaryosu üretimine uygun hale getirilmiştir. Ardından ABC algoritmasının yakınsama hızını iyileştirmek ve değerlendirme sayısını da etkin kullanmak için kapsanan hedeflerin tutulduğu arşiv mekanizması ile iyileştirilmiştir. Son olarak, popülasyonda çeşitliliği artırmak için çeşitli modifikasyonlar gerçekleştirilmiştir. Deneylerin ilk kısmında ise literatürde önerilmiş çaprazlama metotlarının etkinliği incelenmiş ve ABC algoritmasına en uygun çaprazlama metodu belirlenmiştir. Daha sonra arşivin etkinliğini incelemek adına, ABC algoritmasının arşivli ve arşivsiz versiyonları kıyaslanmıştır. Çeşitliliği artırmak adına önerilen MABC algoritmasının sınıfların çeşitli zorluk seviyelerine göre performansı

irdelendikten sonra, literatürde sıkça kullanılan MGA, SGA ve RS algoritmaları ile kıyaslanıp istatistiksel analizleri sunulmuştur. Son olarak önerilen algoritmaların çeşitlilik ve yakınsama analizleri gerçekleştirilmiştir. Deneyler SF110 corpus problem seti üzerinde gerçekleştirilmiş olup, elde edilen sonuçlara göre SPR metodunun ABC algoritması için en uygun çaprazlama yöntemi olduğu belirlenmiştir. Arşiv mekanizmasının da ABC algoritmasının performansına oldukça olumlu etkileri olduğu belirlenmiştir. Önerilen değişiklikler algoritmanın keşif ve faydalanma yeteneklerini kuvvetlendirmiştir. MABC algoritmasının dezavantajı ise yeni birey üretim ve havuz mekanizmalarında çok fazla sistem kaynağı harcadığı için diğer algoritmalarından daha fazla sistem kaynağına ihtiyaç duymasıdır.

Mikroservis mimarisi, bir uygulamayı yüksek düzeyde bakım yapılabilir servisler topluluğuna dönüştürür ve monolitik mimarinin karmaşıklığını azaltır ancak Otomatik Hizmet Keşfi ve Aşırı Geç Bağlanma gibi özellikleri, entegrasyon testini engellediği ve bir servisin kullanım başına maliyeti de birim testini ve QoS testini etkilediği için bu mimariyi test etmek zordur. Uygulamaların her geçen gün daha da büyüdüğü günümüzde mikroservis mimarisini kullanan kurum sayısı artmaktadır ve bu uygulamaların test edilmesi daha da önemli hale gelmektedir. Bu tez çalışmasında, mikroservis mimarisinin yapı taşlarından olan RESTful web servislerin testi için ABC algoritması uygun hale getirilmiştir ve ayrıca eksiklikleri giderilerek Hiper-kâşif Arı Fazlı Dinamik Yapay Arı Koloni Algoritması (DABC-HS) algoritması önerilmiştir. Hiper-kâşif arı fazı eklemesiyle birlikte ABC algoritmasının keşif yeteneği artırılmıştır. Ayrıca yeni önerilen algorithma, dinamik görevli/gözcü arı sayısı keşif ve faydalanma arasındaki dengeyi kurmada yardımcı olmuştur. İlk olarak ABC ve DABC-HS algoritmalarının kontrol parametreleri belirlenmiş olup ardından bu algoritmaların performansını kapsanan hedefler, bulunan hatalar, üretilen test sayısı ve çalışma zamanı açısından incelemek için literatürdeki problemlerden olan ve 23 binden fazla satır koda sahip catwatch, features-service, proxyprint, rest-ncs, rest-news, rest-scs ve scout-api SUT'ları üzerinde test senaryosu üretimi gerçekleştirilmiş ve elde edilen sonuçlar literatürdeki algoritmalarından olan RS, WTS, MIO ve MOSA'nın almış olduğu sonuçlar ile karşılaştırılmıştır.

Deneyisel çalışmalar DABC-HS algoritmasının en fazla hedef kapsayan algoritma olduğunu, yedi problemin dördünde diğer algoritmaları geride bıraktığını göstermiştir.

Geri kalan üç problemde ise diğer algoritmalara yakın sonuç elde etmiştir. Ayrıca problemler üzerinde toplam 84 hata bulmuştur. DABC-HS çalışma zamanına göre RS kadar hızlı olmasa da WTS ile benzer bir hıza sahiptir ve diğer çok amaçlı algoritmalarından daha hızlı sonuç üretebilmektedir. Test senaryosu üretimi oldukça maliyetli bir eylem olduğu için çalışma hızının yüksek olması oldukça önemlidir. Önerilen algoritmanın genel olarak iyi sonuç üretmesi sebebiyle diğer kombinasyonel problemler için de önemli bir potansiyel taşımaktadır. Ayrıca endüstride bu alanda yapılacak olan çalışmalara ihtiyacın her geçen gün daha da artmasına rağmen literatürde RESTful web servisler için test senaryosu üretimi çalışmaları oldukça kısıtlıdır. Bu nedenle bu çalışmanın literatüre katkısı oldukça önemlidir.

Gerçekleştirilen çalışmalar genel olarak değerlendirilecek olursa, nesne yönelimli programlama için özelleştirilmiş ve çeşitli modifikasyonlarla keşif ve faydalanma yeteneklerinin artırıldığı MABC algoritması çok kriterli bir algoritmadır. Amaç fonksiyonu hesaplamasında birden fazla kriterin toplanması ile aynı anda bütün kriterlerde yüksek kapsamaya ulaşılmaya çalışılmaktadır. Literatürdeki ilk yaklaşımlar da birden fazla kriteri aynı amaç fonksiyonu ile optimize etmeye odaklanmıştır. Önerilen MABC algoritması da bu algoritmalarından daha iyi sonuç üreten başarılı bir algoritmadır. Yazılım testi alanındaki çalışmaların ilerlemesi ile çok amaçlı optimizasyon problemleri ile test senaryosu üretim probleminin çözümü popüler hale gelmiştir ve geliştirilen algoritmalar (MIO, MOSA) çok amaçlı yaklaşımı uygulamıştır. Çok amaçlı algoritmaların elde ettiği sonuçlara göre, çok kriterli algoritmalarından daha iyi performans sergilerken, çalışma hızı olarak geride kalmıştır. Son zamanlarda oldukça popüler hale gelen RESTful test senaryosu üretimi için önerilen çok amaçlı optimizasyon algoritması olan DABC-HS ise literatürdeki diğer algoritmalarından (MIO, MOSA) çok daha iyi sonuç üretirken, çok kriterli bir algoritma olan WTS ile benzer çalışma hızına sahiptir. Yazılım testinin hem zaman alan, hem de ekonomik boyutu oldukça büyük olan bir eylem olması sebebiyle, DABC-HS algoritması bu üstünlüklerinin getirdiği avantaj ile yazılım test senaryosu üretiminde oldukça önemli bir yere sahiptir. DABC-HS algoritmasının literatürde kullanılması ile çok daha fazla hatanın otomatik olarak tespit edilmesi ve yüksek kapsamlı test senaryolarının kısa süre içerisinde oluşturulması sağlanabilmektedir.

Bu deneysel çalışmaların ise eksik kaldığı birkaç nokta bulunmaktadır. Bunlardan biri,

kontrol parametrelerinin yapay problemler ile ayarlanmasıdır. Bu durum, çalışmaların performansını düşüren etmenlerden biridir ve yeterli hesaplama zamanı olduğu durumlarda probleme özel kontrol parametresi ayarlamak algoritmanın performansını artıracaktır. Ayrıca bu tür algoritmaların performansı rastgelelikten etkilenebilmektedir. Bu nedenle 30 farklı deney gerçekleştirilip istatistiksel analizlerle destekleyerek sonuçlar sunulmuştur.

5.2. Öneriler

Yazılım test senaryosu üretim alanı hâlâ araştırılması gereken açık bir alandır. Her ne kadar önerilen algoritmalar literatürde oldukça iyi sonuçlar elde ediyor olsa da yazılımların oldukça hızlı geliştiği günümüz dünyasında daha iyi sonuç üreten ve daha hızlı algoritmalara ihtiyaç bulunmaktadır.

Yazılım test senaryosu üretiminde otomatik programlama yöntemlerinin etkin olarak kullanılabileceği düşünülmektedir. Bu yüzden probleme özgü otomatik programlama yöntemleri geliştirilmelidir.

İlk olarak önerilen algoritma olan MABC algoritmasının çalışma hızı diğer algoritmaların gerisinde kalmıştır. Mevcut mekanizmalar algoritmanın performansını oldukça artırmaktadır fakat aynı mekanizmaların çalışma zamanı açısından daha efektif versiyonlarının araştırılması ve geliştirilmesi ile algoritmanın performansının çok daha iyi olabileceği düşünülmektedir.

Ayrıca meta-sezgisel algoritmaların kontrol parametrelerinin ayarlanması daha iyi performans elde edebilmek açısından oldukça önemlidir. Nesne yönelimli programlama için gerekli çekirdek hesaplama zamanı yaklaşık 172.5 ay iken, RESTful web servis için test senaryosu üretiminde gerekli çekirdek hesaplama zamanı 595 gündür. Zaman açısından çok yüksek gereksinim duyduğu için bu konu üzerine ayrıca çalışmak gerekmektedir.

Çok kriterli MABC algoritmasının amaç fonksiyon toplamında kullanılan w_i faktörü literatürde belirtildiği gibi bir (1) olarak seçilmiştir. Ancak bu faktörün ayarlanması da bir başka optimizasyon problemidir. Doğru katsayılar ile daha yüksek kapsama miktarına

erişilebileceği düşünülmektedir.

Nesne yönelimli programlama için test senaryosu üretimi Java programlama dili için gerçekleştirilmiştir. Gelecek çalışmalarda daha kapsamlı programlama dili destekleri sunulması düşünülmektedir. Nesne yönelimli programlama için test senaryosu üretiminde 6569 sınıf üzerinden elde edilen sonuçlar kıyaslanmıştır. İlerideki çalışmalarda daha kapsamlı sınıf üzerinde analizler gerçekleştirilebilir.

Web servisler için test senaryosu üretimi yalnızca RESTful web servisler için gerçekleşmektedir. Günümüzde yeni yeni popüler olmaya başlayan GraphQL sorgu dili desteği gelecekte sağlanmalıdır.

RESTful web servislerde bazı durumlarda 404-Sayfa Bulunamadı durum kodu yerine 500 durum kodu ile karşılaşılabilmektedir ve bu durum aslında bir hata değildir. Genellikle geliştirici bir kaydın bulunup bulunmadığını kontrol etmediği durumlarda, geçersiz bir id kullanılarak veri çekerken bu durum kodu ile karşılaşılmaktadır. Bu durumda kullanılan "doğrulayıcı" tanımı beklenildiği gibi çalışmamaktadır. Bu durumları da dikkate alarak yeni bir "doğrulayıcı" tanımlamak hata tespitinde oldukça önemlidir ve bu konuda yapılacak olan çalışmalara ihtiyaç bulunmaktadır.

RESTful test senaryosu üretiminde amaç fonksiyonunun ödüllendirdiği her yeni 5xx kodu durumuna sebep olan test senaryoları arşivde saklanmaktadır. Ancak bu tür testler ile ne yapılacağı tam olarak belli değildir. Bir 5xx durum kodu her zaman bir hatanın belirtisi değildir ancak kullanıcıya çıktı olarak 5xx durum kodu elde eden test senaryoları verildiğinde ilgili testin başarısız olmasına sebebiyet verebileceği için pek çok durumda diğer test senaryolarından daha değerlidir ve bu test senaryosunun gerçek çıktısının 5xx olduğu düşünülerek kullanıcıya test senaryosu olarak sunulur. Ancak bu tür bilgilerin kullanıcıya en iyi nasıl iletileceği (test senaryosu önceliklendirilmesi, özel test adları vs.) gelecekte araştırılması gereken konulardandır.

KAYNAKLAR

1. Cerpa, N. and Verner, J.M., 2009. Why did your project fail?, **Communications of the ACM**, **52**(12):130–134.
2. Standish, G., 1994. The CHAOS report. The Standish Group International, **Inc**, **Boston**.
3. Rubinstein, D., 2007. Standish group report: There's less development chaos today, **Software Development Times**, **1**.
4. Glass, R.L., 1998. Software Runaways: Monumental Software Disasters, Prentice-Hall, Inc., USA.
5. Beizer, B., 1990. Software Testing Techniques (2nd Ed.), Van Nostrand Reinhold Co., New York, NY, USA.
6. Rauf, A. and Ramzan, M., 2017. Parallel testing and coverage analysis for context-free applications, **Cluster Computing**, **21**(1):729–739.
7. Ali, S., Briand, L.C., Hemmati, H., and Panesar-Walawege, R.K., 2010. A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation, **IEEE Transactions on Software Engineering**, **36**(6):742–762.
8. Arcuri, A. and Yao, X., 2007. On Test Data Generation of Object-Oriented Software, *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION*, 72–76.
9. Canfora, G. and Di Penta, M., 2009. Service-Oriented Architectures Testing: A Survey, 78–105, Springer Berlin Heidelberg, Berlin, Heidelberg.
10. Bozkurt, M., Harman, M., and Hassoun, Y., 2013. Testing and verification in service-oriented architecture: a survey, **Software Testing, Verification and Reliability**, **23**(4):261–313.

11. Bertolino, A., 2007. Software testing research: Achievements, challenges, dreams, *Future of Software Engineering (FOSE'07)*, 85–103, IEEE.
12. Young, M. and Pezze, M., 2005. Software Testing and Analysis: Process, Principles and Techniques, John Wiley & Sons, Inc., Hoboken, NJ, USA.
13. Zhu, H., Hall, P.A.V., and May, J.H.R., 1997. Software unit test coverage and adequacy, **ACM Computing Surveys**, **29**(4):366–427.
14. Anand, S., Burke, E.K., Chen, T.Y., Clark, J., Cohen, M.B., Grieskamp, W., Harman, M., Harrold, M.J., and Mcminn, P., 2013. An Orchestrated Survey on Automated Software Test Case Generation, **Journal of Systems and Software**, **86**(8):1978–2001.
15. King, J.C., 1975. A New Approach to Program Testing, *Proceedings of the International Conference on Reliable Software*, 228–233, Association for Computing Machinery, New York, NY, USA.
16. Cadar, C., Dunbar, D., and Engler, D.R., 2008. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs, *8th USENIX Symposium on Operating Systems Design and Implementation*, volume 8, 209–224.
17. Khurshid, S., Pasareanu, C.S., and Visser, W., 2003. Generalized Symbolic Execution for Model Checking and Testing, *H. Garavel and J. Hatcliff, Eds., Tools and Algorithms for the Construction and Analysis of Systems*, 553–568, Springer Berlin Heidelberg, Berlin, Heidelberg.
18. Pasareanu, C.S. and Rungta, N., 2010. Symbolic PathFinder: Symbolic Execution of Java Bytecode, **25th IEEE/ACM International Conference on Automated Software Engineering**, **2**:179–180.
19. Sen, K. and Agha, G., 2006. CUTE and jCUTE: Concolic Unit Testing and Explicit Path Model-Checking Tools, **Computer Aided Verification**, **4144**:419–423.
20. Jayaraman, K., Harvison, D., Ganesh, V., and Kiezun, A., 2009. jFuzz: A Concolic Whitebox Fuzzer for Java, *NASA Formal Methods*, 121–125.

21. Kähkönen, K., Launiainen, T., Saarikivi, O., Kauttio, J., Heljanko, K., and Niemelä, I., 2011. LCT: An open source concolic testing tool for Java programs, *The Proceedings of the 6th Workshop on Bytecode Semantics, Verification, Analysis and Transformation*, 75–80.
22. Sen, K., Marinov, D., and Agha, G., 2005. CUTE: A Concolic Unit Testing Engine for C, *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE-13, 263–272, ACM, New York, NY, USA.
23. Chipounov, V., Kuznetsov, V., and Candea, G., 2011. S2E: A Platform for in-Vivo Multi-Path Analysis of Software Systems, *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, 265–278, Association for Computing Machinery, New York, NY, USA.
24. Tillmann, N. and de Halleux, J., 2008. Pex: White Box Test Generation for .NET, **Proc. TAP**, 134–153.
25. de Moura, L. and Bjørner, N., 2008. Z3: An Efficient SMT Solver, *Tools and Algorithms for the Construction and Analysis of Systems*, 337–340, Springer Berlin Heidelberg.
26. Dutertre, B. and De Moura, L., 2006. A fast linear-arithmetic solver for DPLL (T), *International Conference on Computer Aided Verification*, 81–94, Springer.
27. Ganesh, V. and Dill, D.L., 2007. A Decision Procedure for Bit-Vectors and Arrays, **Computer Aided Verification**, 519–531.
28. Huima, A., 2007. Implementing Conformiq Qtronic, A. Petrenko, M. Veanes, J. Tretmans, and W. Grieskamp, Eds., *Testing of Software and Communicating Systems*, 1–12, Springer Berlin Heidelberg, Berlin, Heidelberg.
29. Legeard, B. and Utting, M., 2010. Model-based testing-next generation functional software testing, **SoftwareTech News**, **12**(4):9–18.

30. Campbell, C., Grieskamp, W., Nachmanson, L., Schulte, W., Tillmann, N., and Veanes, M., 2005. Model-based testing of object-oriented reactive systems with Spec Explorer, Report, Technical Report MSR-TR-2005-59, Microsoft Research.
31. Krenn, W., Schlick, R., Tiran, S., Aichernig, B., Jobstl, E., and Brandl, H., 2015. MoMut::UML Model-Based Mutation Testing for UML, *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, IEEE.
32. Kuhn, D.R., Kacker, R.N., and Lei, Y., 2010. Advanced combinatorial test methods for system reliability, **2010 Annual Technical Report of the IEEE Reliability Society, Jan. 2011.**
33. Cochran, W.G. and Designs, G.M.C.E., 1957. John Wiley and Sons, **New York**, 293–316.
34. Fisher, R.A., 1990. The Design of Experiments, Oxford University Press, Oxford.
35. Mandl, R., 1985. Orthogonal Latin squares: an application of experiment design to compiler testing, **Communications of the ACM**, **28**(10):1054–1058.
36. Chen, T.Y., Kuo, F.C., Merkel, R.G., and Tse, T.H., 2010. Adaptive Random Testing: The ART of test case diversity, **Journal of Systems and Software**, **83**(1):60–66.
37. Chen, T.Y., Leung, H., and Mak, I.K., 2005. Adaptive Random Testing, *M.J. Maher, Ed.; Advances in Computer Science - ASIAN 2004. Higher-Level Decision Making*, 320–329, Springer Berlin Heidelberg, Berlin, Heidelberg.
38. White, L.J. and Cohen, E.I., 1980. A domain strategy for computer program testing, **IEEE Transactions on Software Engineering**, **SE-6**(3):247–257.
39. Chan, K.P., Chen, T.Y., and Towey, D., 2006. Restricted Random Testing : Adaptive Random, **International Journal of Software Engineering and Knowledge Engineering**, **16**(4):553–584.

40. Ciupa, I., Leitner, A., Oriol, M., and Meyer, B., 2008. Artoo, *2008 ACM/IEEE 30th International Conference on Software Engineering*, 71–80, IEEE.
41. Liu, H., Xie, X., Yang, J., Lu, Y., and Chen, T.Y., 2010. Adaptive Random Testing by Exclusion through Test Profile, *2010 10th International Conference on Quality Software*, 92–101.
42. Mayer, J., 2005. Lattice-Based Adaptive Random Testing, *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, ASE '05*, 333–336, Association for Computing Machinery, New York, NY, USA.
43. Shahbazi, A., Tappenden, A.F., and Miller, J., 2013. Centroidal voronoi tessellations-a new approach to random testing, **IEEE Transactions on Software Engineering**, **39**(2):163–183.
44. Pacheco, C. and Ernst, M.D., 2007. Randoop: Feedback-Directed Random Testing for Java, *Companion to the 22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications Companion, OOPSLA '07*, 815–816, Association for Computing Machinery, New York, NY, USA.
45. Simons, A.J.H., 2007. JWalk: a tool for lazy, systematic testing of java classes by design introspection and user interaction, **Automated Software Engineering**, **14**(4):369–418.
46. Ma, L., Artho, C., Zhang, C., Sato, H., Gmeiner, J., and Ramler, R., 2015. GRT: Program-Analysis-Guided Random Testing (T), *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE.
47. Prasetya, I.S.W.B., 2015. T3i: a tool for generating and querying test suites for Java, *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*, ACM Press.
48. Csallner, C. and Smaragdakis, Y., 2004. JCrasher: an automatic robustness tester for Java, **Software: Practice and Experience**, **34**(11):1025–1050.
49. Harman, M. and Jones, B.F., 2001. Search-based software engineering, **Information and Software Technology**, **43**(14):833 – 839.

50. Miller, W. and Spooner, D.L., 1976. Automatic Generation of Floating-Point Test Data, **IEEE Transactions on Software Engineering**, **SE-2**(3):223–226.
51. Korel, B., 1990. Automated Software Test Data Generation, **IEEE Transactions on Software Engineering**, **16**(8):870–879.
52. Korel, B., Wedde, H., and Ferguson, R., 1992. Dynamic method of test data generation for distributed software, **Information and Software Technology**, **34**(8):523–531.
53. Xanthakis, S., Ellis, C., Skourlas, C., Gal, A.L., Katsikas, S., and Karapoulis, K., 1992. Application of Genetic Algorithms to Software Testing.
54. Fraser, G. and Arcuri, A., 2013. Evosuite: On the challenges of test case generation in the real world, *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, 362–369, IEEE.
55. Sakti, A., Pesant, G., and Guéhéneuc, Y.G., 2015. Instance Generator and Problem Representation to Improve Object Oriented Code Coverage, **IEEE Transactions on Software Engineering**, **41**(3):294–313.
56. Arcuri, A., 2018. EvoMaster: Evolutionary Multi-context Automated System Test Generation, *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, 394–397.
57. McMinn, P., 2004. Search-based Software Test Data Generation: A Survey: Research Articles, **Software Testing, Verification and Reliability**, **14**(2):105–156.
58. Afzal, W., Torkar, R., and Feldt, R., 2009. A Systematic Review of Search-based Testing for Non-functional System Properties, **Information and Software Technology**, **51**(6):957–976.
59. Rähkä, O., 2010. A survey on search-based software design, **Computer Science Review**, **4**(4):203 – 249.

60. McMinn, P., 2011. Search-Based Software Testing: Past, Present and Future, *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*, 153–163.
61. Harman, M., Mansouri, S.A., and Zhang, Y., 2009. Search based software engineering: A comprehensive analysis and review of trends techniques and applications, Technical report, Department of Computer Science, King's College London.
62. Harman, M., Mansouri, S.A., and Zhang, Y., 2012. Search-based Software Engineering: Trends, Techniques and Applications, **ACM Comput. Surv.**, **45**(1):11:1–11:61.
63. Panichella, A., 2014. Search-Based Software Maintenance and Testing, Doktora Tezi, Università degli studi di Salerno.
64. Baresel, A., Binkley, D., Harman, M., and Korel, B., 2004. Evolutionary Testing in the Presence of Loop-Assigned Flags: A Testability Transformation Approach, **SIGSOFT Software Engineering Notes**, **29**(4):108–118.
65. Baresel, A., Sthamer, H., and Schmidt, M., 2002. Fitness function design to improve evolutionary structural testing, *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, 1329–1336.
66. Bottaci, L., 2002. Instrumenting programs with flag variables for test data search by genetic algorithm, *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, 1337–1342.
67. Harman, M., Hu, L., Hierons, R., Wegener, J., Sthamer, H., Baresel, A., and Roper, M., 2004. Testability transformation, **IEEE Transactions on Software Engineering**, **30**(1):3–16.
68. Yoo, S. and Harman, M., 2012. Regression testing minimization, selection and prioritization: a survey, **Software Testing, Verification and Reliability**, **22**(2):67–120.

69. Do, H., Mirarab, S., Tahvildari, L., and Rothermel, G., 2010. The effects of time constraints on test case prioritization: A series of controlled experiments, **IEEE Transactions on Software Engineering**, **36**(5):593–617.
70. Harrold, M.J., Gupta, R., and Soffa, M.L., 1993. A methodology for controlling the size of a test suite, **ACM Transactions on Software Engineering and Methodology (TOSEM)**, **2**(3):270–285.
71. Bates, S. and Horwitz, S., 1993. Incremental program testing using program dependence graphs, *Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 384–396.
72. Offutt, J., Pan, J., and Voas, J.M., 1995. Procedures for reducing the size of coverage-based test sets, *Proceedings of the 12th International Conference on Testing Computer Software*, 111–123, ACM Press New York.
73. Rothermel, G., Harrold, M.J., Von Ronne, J., and Hong, C., 2002. Empirical studies of test-suite reduction, **Software Testing, Verification and Reliability**, **12**(4):219–249.
74. Black, J., Melachrinoudis, E., and Kaeli, D., 2004. Bi-criteria models for all-uses test suite reduction, *Proceedings. 26th International Conference on Software Engineering*, 106–115, IEEE.
75. McMaster, S. and Memon, A.M., 2005. Call stack coverage for test suite reduction, *21st IEEE International Conference on Software Maintenance (ICSM'05)*, 539–548, IEEE.
76. Jeffrey, D. and Gupta, N., 2005. Test suite reduction with selective redundancy, *21st IEEE International Conference on Software Maintenance (ICSM'05)*, 549–558, IEEE.
77. Yoo, S., Harman, M., and Ur, S., 2011. Highly scalable multi objective test suite minimisation using graphics cards, *International Symposium on Search Based Software Engineering*, 219–236, Springer.

78. Harris, P. and Raju, N., 2015. A Greedy Approach for Coverage-Based Test Suite Reduction., **International Arab Journal of Information Technology**, **12**(1).
79. Fischer, K., 1977. A test case selection method for the validation of software maintenance modifications, *COMPSAC, (New York)*, 421–426, IEEE.
80. Elbaum, S., Malishevsky, A., and Rothermel, G., 2001. Incorporating varying test costs and fault severities into test case prioritization, *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, 329–338, IEEE.
81. Malishevsky, A.G., Ruthruff, J.R., Rothermel, G., and Elbaum, S., 2006. Cost-cognizant test case prioritization, Technical report, Technical Report TR-UNL-CSE-2006-0004, University of Nebraska-Lincoln.
82. Walcott, K.R., Soffa, M.L., Kapfhammer, G.M., and Roos, R.S., 2006. Timeaware test suite prioritization, *Proceedings of the 2006 international symposium on Software testing and analysis*, 1–12.
83. Yoo, S. and Harman, M., 2007. Pareto efficient multi-objective test case selection, *Proceedings of the 2007 international symposium on Software testing and analysis*, 140–150.
84. Elbaum, S., Malishevsky, A.G., and Rothermel, G., 2000. Prioritizing test cases for regression testing, *Proceedings of the 2000 ACM SIGSOFT international symposium on Software testing and analysis*, 102–112.
85. Mondal, S. and Nasre, R., 2021. Hansie: Hybrid and consensus regression test prioritization, **Journal of Systems and Software**, **172**:110850.
86. Roper, M., 1997. Computer aided software testing using genetic algorithms, *10th International Quality Week*, USA.
87. Watkins, A.L., 1995. The automatic generation of test data using genetic algorithms, *Proceedings of the 4th software quality conference*, volume 2, 300–309.
88. Wappler, S. and Lammermann, F., 2005. Using evolutionary algorithms for the unit testing of object-oriented software, *Proceedings of the 2005 conference on Genetic and evolutionary computation*, 1053, ACM Press.

89. Fraser, G. and Arcuri, A., 2011. Evosuite: automatic test suite generation for object-oriented software, *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 416–419.
90. Fraser, G. and Arcuri, A., 2012. Whole test suite generation, **IEEE Transactions on Software Engineering**, **39**(2):276–291.
91. Pargas, R.P., Harrold, M.J., and Peck, R.R., 1999. Test-data generation using genetic algorithms, **Software Testing, Verification and Reliability**, **9**(4):263–282.
92. Shamshiri, S., Rojas, J.M., Fraser, G., and McMin, P., 2015. Random or genetic algorithm search for object-oriented test suite generation?, *Proceedings of the 2015 annual conference on genetic and evolutionary computation*, 1367–1374.
93. Tracey, N.J., 2000. A search-based automated test-data generation framework for safety-critical software, Doktora Tezi, Citeseer.
94. Wegener, J., Baresel, A., and Sthamer, H., 2001. Evolutionary test environment for automatic structural testing, **Information and Software Technology**, **43**(14):841–854.
95. De Jesus Guimaraes Oliveira, C.E., 2019. Mapping the Effectiveness of Automated Test Suite Generation Techniques, Doktora Tezi, Monash University.
96. Aleti, A. and Grunske, L., 2015. Test data generation with a Kalman filter-based adaptive genetic algorithm, **Journal of Systems and Software**, **103**:343–352.
97. Watkins, A. and Hufnagel, E.M., 2006. Evolutionary test data generation: a comparison of fitness functions, **Software: Practice and Experience**, **36**(1):95–116.
98. Tracey, N., Clark, J.A., and Mander, K., 1998. The way forward for unifying dynamic test-case generation: The optimisation-based approach, *Proceedings of the IFIP International Workshop on Dependable Computing and Its Applications (DCIA)*, York.

99. Tracey, N., Clark, J., Mander, K., and McDermid, J., 1998. An automated framework for structural test-data generation, *Proceedings 13th IEEE International Conference on Automated Software Engineering (Cat. No. 98EX239)*, 285–288, IEEE.
100. Alshraideh, M. and Bottaci, L., 2006. Search-based software test data generation for string data using program-specific search operators, **Software Testing, Verification and Reliability**, **16**(3):175–203.
101. Jones, B.F., Sthamer, H.H., and Eyres, D.E., 1996. Automatic structural testing using genetic algorithms, **Software Engineering Journal**, **11**(5):299–306.
102. Tonella, P., 2004. Evolutionary Testing of Classes, **SIGSOFT Softw. Eng. Notes**, **29**(4):119–128.
103. Sahin, O. and Akay, B., 2016. Comparisons of Metaheuristic Algorithms and Fitness Functions on Software Test Data Generation, **Applied Soft Computing**, **49**(C):1202–1214.
104. Harman, M., Mansouri, S.A., and Zhang, Y., 2012. Search-based Software Engineering: Trends, Techniques and Applications, **ACM Computing Surveys**, **45**(1):11:1–11:61.
105. Harman, M., McMinn, P., de Souza, J., and Yoo, S., 2012. Search Based Software Engineering: Techniques, Taxonomy, Tutorial, B. Meyer and M. Nordio, Eds:, Empirical Software Engineering and Verification, volume 7007 of Lecture Notes in Computer Science, 1–59, Springer Berlin Heidelberg.
106. Khari, M. and Kumar, P., 2017. An extensive evaluation of search-based software testing: a review, **Soft Computing**, **23**(6):1933–1946.
107. Windisch, A., Wappler, S., and Wegener, J., 2007. Applying Particle Swarm Optimization to Software Testing, *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO '07*, 1121–1128, ACM, New York, NY, USA.

108. Tiwari, S., Mishra, K., and Misra, A., 2013. Test Case Generation for Modified Code Using a Variant of Particle Swarm Optimization (PSO) Algorithm, *2013 10th International Conference on Information Technology: New Generations*, Institute of Electrical & Electronics Engineers (IEEE).
109. Zhu, X. and Yang, X., 2010. Software Test Data Generation Automatically Based on Improved Adaptive Particle Swarm Optimizer, *2010 International Conference on Computational and Information Sciences*, 1300–1303.
110. Dahiya, S.S., Chhabra, J.K., and Kumar, S., 2011. PSO Based Pseudo Dynamic Method for Automated Test Case Generation Using Interpreter, *Proceedings of the Second International Conference on Advances in Swarm Intelligence - Volume Part I*, ICSI'11, 147–156, Springer-Verlag, Berlin, Heidelberg.
111. Sanjay Singla, Dharminder Kumar, H.M.R. and Singla, P., 2011. A Hybrid PSO Approach to Automate Test Data Generation for Data Flow Coverage with Dominance Concepts, **International Journal of Advanced Science and Technology**, 37.
112. Latiu, G.I., Cret, O.A., and Vacariu, L., 2012. Automatic Test Data Generation for Software Path Testing Using Evolutionary Algorithms, *Proceedings of the 2012 Third International Conference on Emerging Intelligent Data and Web Technologies*, EIDWT '12, 1–8, IEEE Computer Society, Washington, DC, USA.
113. Becerra, R.L., Sagarna, R., and Yao, X., 2009. An evaluation of Differential Evolution in software test data generation, *2009 IEEE Congress on Evolutionary Computation*, Institute of Electrical & Electronics Engineers (IEEE).
114. Jianfeng, W., Changan, W., and Shouda, J., 2013. Test Data Generation Algorithm of Combinatorial Testing Based on Differential Evolution, *2013 Third International Conference on Instrumentation, Measurement, Computer, Communication and Control*, Institute of Electrical & Electronics Engineers (IEEE).

115. Liang, X., Guo, S., Huang, M., and Jiao, X., 2014. Combinatorial Test Case Suite Generation Based on Differential Evolution Algorithm, **JSW**, **9**(6).
116. Mala, D.J. and Mohan, V., 2009. ABC Tester Artificial Bee Colony Optimization For Software Test Suite Optimization, **IJSE International Journal Of Software Engineering**, **2**(2):15–48.
117. Mala, D., Kamalapriya, M., Shobana, R., and Mohan, V., 2009. A non-pheromone based intelligent swarm optimization technique in software test suite optimization, *Intelligent Agent Multi-Agent Systems, 2009. IAMA 2009. International Conference on*, 1–5.
118. Mala, D.J., Mohan, V., and Kamalapriya, M., 2010. Automated software test optimisation framework – an artificial bee colony optimisation-based approach, **IET Software**, **4**:334–348(14).
119. Dahiya, S., Chhabra, J., and Kumar, S., 2010. Application of Artificial Bee Colony Algorithm to Software Testing, *Software Engineering Conference (ASWEC), 2010 21st Australian*, 149–154.
120. Lam, S.S.B., Raju, M.L.H.P., M, U.K., Ch, S., and Srivastav, P.R., 2012. Automated Generation of Independent Paths and Test Suite Optimization Using Artificial Bee Colony, **Procedia Engineering**, **30**:191 – 200, international Conference on Communication Technology and System Design 2011.
121. Malhotra, R. and Khari, M., 2014. Test Suite Optimization Using Mutated Artificial Bee Colony, *Proc. of Int. Conf. on Advances in Communication Network and Computing, CNC*, 45 – 54.
122. Malhotra, R., Anand, C., Jain, N., and Mittal, A., 2014. Article: Comparison of Search based Techniques for Automated Test Data Generation, **International Journal of Computer Applications**, **95**(23):4–8.
123. Suri, D. and Kaur, P., 2014. Path Based Test Suite Augmentation using Artificial Bee Colony Algorithm, **International Journal For Research in Applied Science and Engineering Technology**, **2**:156–164.

124. Srivatsava, P.R., Mallikarjun, B., and Yang, X.S., 2013. Optimal test sequence generation using firefly algorithm, **Swarm and Evolutionary Computation**, **8**:44 – 53.
125. Chen, T.Y., Kuo, F.C., and Liu, H., 2009. Adaptive random testing based on distribution metrics, **Journal of Systems and Software**, **82**(9):1419–1433.
126. Arcuri, A. and Briand, L., 2011. Adaptive random testing: An illusion of effectiveness?, *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, 265–275.
127. Jones, B.F., Eyres, D.E., and Sthamer, H.H., 1998. A strategy for using genetic algorithms to automate branch and fault-based testing, **The Computer Journal**, **41**(2):98–107.
128. Srivastava, P.R. and Kim, T.h., 2009. Application of genetic algorithm in software testing, **International Journal of software Engineering and its Applications**, **3**(4):87–96.
129. Aleti, A., Moser, I., and Grunske, L., 2017. Analysing the fitness landscape of search-based software testing problems, **Automated Software Engineering**, **24**(3):603–621.
130. Harman, M. and McMinn, P., 2009. A theoretical and empirical study of search-based testing: Local, global, and hybrid search, **IEEE Transactions on Software Engineering**, **36**(2):226–247.
131. McMinn, P., Harman, M., Lakhotia, K., Hassoun, Y., and Wegener, J., 2011. Input domain reduction through irrelevant variable removal and its effect on local, global, and hybrid search-based structural test data generation, **IEEE Transactions on Software Engineering**, **38**(2):453–477.
132. Michael, C.C., McGraw, G., and Schatz, M.A., 2001. Generating software test data by evolution, **IEEE Transactions on Software Engineering**, **27**(12):1085–1110.

133. Wegener, J. and Bühler, O., 2004. Evaluation of different fitness functions for the evolutionary testing of an autonomous parking system, *Genetic and Evolutionary Computation Conference*, 1400–1412, Springer.
134. Wegener, J. and Grochtman, M., 1998. Verifying timing constraints of real-time systems by means of evolutionary testing, **Real-Time Systems**, **15**(3):275–298.
135. Derderian, K., Hierons, R.M., Harman, M., and Guo, Q., 2006. Automated unique input output sequence generation for conformance testing of FSMs, **The Computer Journal**, **49**(3):331–344.
136. Colanzi, T.E., Assunção, W.K.G., Vergilio, S.R., and Pozo, A., 2011. Integration test of classes and aspects with a multi-evolutionary and coupling-based approach, *International Symposium on Search Based Software Engineering*, 188–203, Springer.
137. Briand, L.C., Feng, J., and Labiche, Y., 2002. Using genetic algorithms and coupling measures to devise optimal integration test orders, *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, 43–50.
138. Harman, M., Jia, Y., and Langdon, W.B., 2011. Strong higher order mutation-based test data generation, *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 212–222.
139. Zhan, Y. and Clark, J.A., 2005. Search-based mutation testing for simulink models, *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, 1061–1068.
140. Yoo, S., Harman, M., Tonella, P., and Susi, A., 2009. Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge, *Proceedings of the eighteenth international symposium on Software testing and analysis*, 201–212.
141. Del Grosso, C., Antoniol, G., Di Penta, M., Galinier, P., and Merlo, E., 2005. Improving network applications security: a new heuristic to generate

- stress testing data, *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, 1037–1043.
142. Alshahwan, N. and Harman, M., 2011. Automated web application testing using search based software engineering, *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, 3–12, IEEE.
 143. Arcuri, A., 2019. RESTful API Automated Test Case Generation with EvoMaster, **ACM Transactions on Software Engineering and Methodology**, **28**(1):1–37.
 144. Wappler, S. and Wegener, J., 2006. Evolutionary unit testing of object-oriented software using strongly-typed genetic programming, *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 1925–1932, ACM.
 145. Wappler, S. and Wegener, J., 2006. Evolutionary unit testing of object-oriented software using a hybrid evolutionary algorithm, *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, 851–858, IEEE.
 146. Liaskos, K., Roper, M., and Wood, M., 2007. Investigating data-flow coverage of classes using evolutionary algorithms, *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, 1140, ACM.
 147. Sagarna, R., Arcuri, A., and Yao, X., 2007. Estimation of distribution algorithms for testing object oriented software, *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, 438–444, IEEE.
 148. Wappler, S. and Schieferdecker, I., 2007. Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering - ASE07, chapter Improving evolutionary class testing in the presence of non-public methods, 381–384, ACM Press.
 149. Fraser, G. and Zeller, A., 2012. Mutation-Driven Generation of Unit Tests and Oracles, **IEEE Transactions on Software Engineering**, **38**(2):278–292.
 150. Fraser, G. and Arcuri, A., 2015. Achieving scalable mutation-based generation of whole test suites, **Empirical Software Engineering**, **20**(3):783–812.

151. Campos, J., Ge, Y., Fraser, G., Eler, M., and Arcuri, A., 2017. An Empirical Evaluation of Evolutionary Algorithms for Test Suite Generation, *T. Menzies and J. Petke, Eds., Search Based Software Engineering*, 33–48, Springer International Publishing, Cham.
152. Rojas, J.M., Vivanti, M., Arcuri, A., and Fraser, G., 2017. A detailed investigation of the effectiveness of whole test suite generation, **Empirical Software Engineering**, **22**(2):852–893.
153. Shahabi, M.M.D., Badiei, S.P., Beheshtian, S.E., Akbari, R., and Moosavi, S.M.R., 2017. On the performance of EvoPSO: A PSO based algorithm for test data generation in EvoSuite, *2017 2nd Conference on Swarm Intelligence and Evolutionary Computation (CSIEC)*, 129–134.
154. Mehdi, M., Parsa, S., Ehsan, S., Akbari, R., and Mohammad, S., 2017. EVOTLBO: A TLBO based Method for Automatic Test Data Generation in EvoSuite, **International Journal of Advanced Computer Science and Applications**, **8**(6).
155. Segura, S., Parejo, J.A., Troya, J., and Ruiz-Cortés, A., 2017. Metamorphic testing of RESTful web APIs, **IEEE Transactions on Software Engineering**, **44**(11):1083–1099.
156. Di Penta, M., Canfora, G., Esposito, G., Mazza, V., and Bruno, M., 2007. Search-based testing of service level agreements, *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, 1090–1097, ACM.
157. Arcuri, A., 2017. RESTful API Automated Test Case Generation, *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 9–20.
158. Arcuri, A., 2018. Test suite generation with the Many Independent Objective (MIO) algorithm, **Information and Software Technology**, **104**:195–206.
159. Arcuri, A. and Galeotti, J.P., 2019. SQL data generation to enhance search-based system testing, *Proceedings of the Genetic and Evolutionary Computation*

- Conference on - GECCO '19*, 1390–1398, ACM Press, New York, New York, USA.
160. Zhang, M., Marculescu, B., and Arcuri, A., 2019. Resource-based test case generation for RESTful web services, *Proceedings of the Genetic and Evolutionary Computation Conference on - GECCO '19*, 1426–1434, ACM Press, New York, New York, USA.
 161. Arcuri, A. and Galeotti, J.P., 2020. Handling SQL Databases in Automated System Test Generation, **ACM Transactions on Software Engineering and Methodology**.
 162. Arcuri, A., 2020. Automated Blackbox and Whitebox Testing of RESTful APIs With EvoMaster, **IEEE Software**, 0–0.
 163. Viglianisi, E., Dallago, M., and Ceccato, M., 2020. RESTTESTGEN: Automated Black-Box Testing of RESTful APIs, *Proceedings - 2020 IEEE 13th International Conference on Software Testing, Verification and Validation, ICST 2020*.
 164. Sharma, C.M., Porwal, R., and Sharma, D., 2013. Testing Object Oriented Software: Issues, State-of-the-art and Future, *International Journal of Computer Applications, 8th National Conference on Next generation Computing Technologies and Applications (NGCTA-2013)*, 25–28.
 165. Rojas, J.M., Campos, J., Vivanti, M., Fraser, G., and Arcuri, A., 2015. Combining Multiple Coverage Criteria in Search-Based Unit Test Generation, *M. Barros and Y. Labiche, Eds., Search-Based Software Engineering*, 93–108, Springer International Publishing, Cham.
 166. Ferrante, J., Ottenstein, K.J., and Warren, J.D., 1987. The program dependence graph and its use in optimization, **ACM Transactions on Programming Languages and Systems (TOPLAS)**, **9**(3):319–349.
 167. Ashouri, A.H., Killian, W., Cavazos, J., Palermo, G., and Silvano, C., 2018. A Survey on Compiler Autotuning using Machine Learning, **ACM Computing Surveys**, **51**(5):1–42.

168. Harrold, M.J. and Rothermel, G., 1994. Performing Data Flow Testing on Classes, *Proceedings of the 2Nd ACM SIGSOFT Symposium on Foundations of Software Engineering*, SIGSOFT '94, 154–163, ACM, New York, NY, USA.
169. Son, H.S., Park, Y.B., and Kim, R.Y.C., 2016. MCCFG: an MOF-based multiple condition control flow graph for automatic test case generation, **Cluster Computing**.
170. Karaboga, D., 2005. An Idea Based On Honey Bee Swarm For Numerical Optimization, Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department.
171. Fraser, G. and Arcuri, A., 2011. Evolutionary Generation of Whole Test Suites, *2011 11th International Conference on Quality Software*, 31–40.
172. Fraser, G., Arcuri, A., and McMinn, P., 2015. A Memetic Algorithm for Whole Test Suite Generation, **Journal of Systems and Software**, **103**(C):311–327.
173. Vasconcelos, J., Ramirez, J.A., Takahashi, R., and Saldanha, R., 2001. Improvements in genetic algorithms, **IEEE Transactions on Magnetics**, **37**(5):3414–3417.
174. Arcuri, A., 2012. A theoretical and empirical analysis of the role of test sequence length in software testing for structural coverage, **IEEE Transactions on Software Engineering**, **38**(3):497–519.
175. Arcuri, A., Iqbal, M.Z., and Briand, L., 2010. Formal analysis of the effectiveness and predictability of random testing, *Proceedings of the 19th international symposium on Software testing and analysis*, 219–230.
176. Duran, J.W. and Ntafos, S.C., 1984. An Evaluation of Random Testing, **IEEE Transactions on Software Engineering**, **SE-10**(4):438–444.
177. Fraser, G. and Arcuri, A., 2014. A Large-Scale Evaluation of Automated Unit Test Generation Using EvoSuite, **ACM Transactions on Software Engineering and Methodology**, **24**(2):8:1–8:42.

178. Arcuri, A. and Fraser, G., 2013. Parameter tuning or default values? An empirical investigation in search-based software engineering, **Empirical Software Engineering**, **18**(3):594–623.
179. Panichella, A., Kifetew, F.M., and Tonella, P., 2018. A large scale empirical comparison of state-of-the-art search-based test case generators, **Information and Software Technology**, **104**:236–256.
180. Nadareishvili, I., Mitra, R., McLarty, M., and Amundsen, M., 2016. Microservice architecture: aligning principles, practices, and culture, " O'Reilly Media, Inc."
181. Fielding, R.T. and Taylor, R.N., 2000. Architectural styles and the design of network-based software architectures, Doktora Tezi, University of California.
182. Barr, E.T., Harman, M., McMinn, P., Shahbaz, M., and Yoo, S., 2014. The oracle problem in software testing: A survey, **IEEE Transactions on Software Engineering**, **41**(5):507–525.
183. Fraser, G. and Arcuri, A., 2015. 1600 faults in 100 projects: automatically finding faults while achieving high coverage with evosuite, **Empirical Software Engineering**, **20**(3):611–639.
184. Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T., 2000. A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II, Parallel Problem Solving from Nature PPSN VI, 849–858, Springer Berlin Heidelberg.
185. Panichella, A., Kifetew, F.M., and Tonella, P., 2018. Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets, **IEEE Transactions on Software Engineering**, **44**(2):122–158.

EK-1

EKLER

Tablo EK-1.1. Her bir sınıfın 20 bağımsız koşma üzerinden Dal Kapsamı metriğine göre Wilcoxon test sonuçları.

Sınıf Adı	MABC/ABC	MABC/MGA	MABC/SGA	MABC/RS
accessories.plugins.time.JDayChooser	0.001	0.052	0.005	0.258
apbs_mem_gui.Main	1	1	1	1
bcry.battlecryGUI	1	1	1	0.342
bcry.bcGenerator	0.79	0.435	0.335	0.092
bible.obj.Verse	1	1	1	0
bible.util.Queue	1	1	1	0
bierse.controller.DefaultSettingsController	0.089	0.081	0.081	0.158
bierse.model.Drink	0.797	0.824	0.765	0
Capture	1	1	1	1
ch.bfh.egov.nutzenportfolio.common.AuswertungGrafik	0	0	0	0.73
ch.bfh.egov.nutzenportfolio.service.fragebogen.NaOpNuDaoService	1	1	1	1
client.network.ForwardingObserver	0	0.004	0.012	0.003
com.aELITIS.azureus.ui.swt.feature.FeatureManagerUIListener	0.922	0.896	0.717	0.574
com.aELITIS.azureus.ui.swt.skin.SWTSkinObjectContainer	1	1	1	0
com.allenstudio.ir.ui.MainMenu	1	1	1	1
com.allenstudio.ir.util.XmlElement	0.087	0.297	0.792	0
com.atlassw.tools.eclipse.checkstyle.projectconfig.ProjectConfiguration	1	1	1	1
com.browsersoft.aacs.User	0.48	0.964	0.676	0
com.browsersoft.openhre.hl7.impl.regular.ExpressionImpl	1	1	1	0
com.densebrain.rif.server.transport.WebServiceDescriptor	1	1	1	0
com.eteks.sweethome3d.j3d.Room3D	1	1	1	0.009
com.eteks.sweethome3d.SweetHome3D	0.003	0.002	0.006	0
com.eteks.sweethome3d.viewcontroller.HomeController	0.001	0	0.005	0
com.eteks.sweethome3d.viewcontroller.RoomController	0.342	0.33	1	1
com.gbshape.dbe.sapdb.SapdbTableList	1	1	1	1
com.gbshape.dbe.utils.DBEHelper	0	0	0.066	0.001
com.gotoservers.ftp.FtpApplet	1	1	1	1
com.hf.sfm.crypt.Base64	1	1	1	0.025
com.hf.sfm.util.Loader	0	0.013	0	0.294
com.ib.client.ComboLeg	0.33	1	1	0
com.ib.client.ExecutionFilter	1	1	0.323	0
com.imsmart.servlet.HTMLFilter	1	1	1	1
com.liferay.portlet.documentlibrary.model.DLSyncWrapper	1	1	1	0
com.lts.event.SimpleKeyListenerHelper	1	1	1	0
com.lts.io.DirectoryScanner	0	0.001	0	0
com.lts.swing.combobox.SimpleComboBox	0.381	0.279	0.33	0.324
com.mentorgen.tools.profile.runtime.Profile	0.04	0.111	0.876	0.008
com.pmdesigns.jvc.tools.JVCParser	0.042	0.004	0.069	0.01
com.pmdesigns.jvc.tools.JVCParserTokenManager	0.117	0.223	0.323	0.693
com.pmdesigns.jvc.tools.TokenMgrError	0	0.006	0	0
com.sap.managementconsole.soap.axis.sapcontrol.HeapInfo	0.081	0.395	0.298	0
com.sap.managementconsole.soap.axis.sapcontrol.J2EEApplicationAlias	0.605	0.174	0.162	0
com.soops.CEN4010.JMCA.JMCAAnalyzer	0.74	0.137	0.395	0.47
com.soops.CEN4010.JMCA.JParser.JavaCharStream	0	0	0	0
com.soops.CEN4010.JMCA.JParser.JavaParser	1	1	1	1
com.soops.CEN4010.JMCA.JParser.JavaParserTokenManager	0	0	0	0.051
com.werken.saxpath.XPathLexer	0	0	0	0
com.yahoo.platform.yui.compressor.JavaScriptCompressor	0.723	0.224	0.167	0
com.yahoo.platform.yui.compressor.ScriptOrFnScope	1	1	1	0.004
com.yahoo.platform.yui.compressor.YUICompressor	0.051	0.174	0.311	0.001
Convert	0.605	0.615	0.556	0.174
corina.graph.GrapherPanel	1	1	1	1
corina.graph.StandardPlot	1	1	1	1
corina.site.SiteDB	0.611	0.148	0.276	0.688
de.beiri22.stringincrementor.helper.IndexedString	0.28	0.756	0.659	0.071
de.beiri22.stringincrementor.StringIncrementor	0.111	0.148	0.046	0.006
de.huxhorn.lilith.data.access.AccessEvent	0.003	0.015	0.042	0
de.huxhorn.lilith.swing.preferences.ConditionTableModel	1	1	0.37	0
de.huxhorn.lilith.swing.table.renderer.MethodRenderer	0.446	0.89	0.446	0.02
de.huxhorn.lilith.swing.ViewActions	1	1	1	0
de.outstare.fortbattleplayer.model.impl.SimpleSector	0.232	0.113	0.591	0.225
de.outstare.fortbattleplayer.statistics.BattleStatistics	0.724	0.903	0.011	0
de.paragon.explorer.event.ExplorerFrameEventConverter	1	1	1	1

Tablo EK-1.1. Her bir sınıfın 20 bağımsız koşma üzerinden Dal Kapsamı metriğine göre Wilcoxon test sonuçları.

Sınıf Adı	MABC/ABC	MABC/MGA	MABC/SGA	MABC/RS
de.paragon.explorer.model.AttributeModelComparator	0.862	0.195	0.509	0.002
de.progra.charting.CoordSystemUtilities	0.212	0.033	0.118	0.013
de.progra.charting.DefaultChart	0.035	0.196	0.176	0.126
dsachat.client.gui.InternalChatFrame	1	1	1	1
dsachat.server.Handler	0	0.051	0.001	0.002
edu.uiuc.ndiipp.hubandspoke.profile.HaSMETSWebValidator	1	1	1	1
edu.uiuc.ndiipp.hubandspoke.profile.TestHaSMETSProfile	1	1	1	1
fi.vtt.noen.mfw.bundle.probe.shared.ProbeInformation	0.003	0.001	0.416	0
fi.vtt.noen.testgen.observations.formatter.DaikonFormatter	0.267	0.244	0.504	0.001
fps370.Fps370Panel	1	1	1	1
fr.pingtimeout.jtail.gui.action.IndexFileAction	1	1	1	0.073
fr.pingtimeout.jtail.gui.view.JTailPanel	0.19	0.141	0	0
framework.util.jdbc.JDBCUtils	0.177	0.964	0.528	0
framework.util.StringUtils	0	0.117	0.179	0
freemind.extensions.ExportHook	0.122	0.952	0.407	0.026
geo.google.datamodel.GeoStatusCode	1	1	1	1
geo.google.mapping.AddressToUsAddressFunctor	0	0	0.008	0
ghm.follow.gui.TabbedPane	0.091	0.016	0.227	0.155
ghm.follow.search.SearchableTextPane	0.181	0.789	0.028	0
glengineer.agents.GroupAgent	0.207	0.158	0.098	0.014
glengineer.blocks.Block	1	1	1	0
gui.AudioManager	1	1	1	0
gui.GroupPanel	1	1	1	1
httpanalyzer.HttpAnalyzerView	1	1	1	1
httpanalyzer.Password	1	1	1	1
ioproject.server.network.ClientGroup	0	0.002	0.04	0.002
ipac.BinaryCalculate	0.001	0.086	0.001	0.002
ipac.WhoIS	0.502	0.475	0.423	0.046
jahuwaldt.plot.PlotAxis	0.511	0.877	0.574	0
jav.entrada.Abrir	1	1	1	1
jcmdline.HelpCmdLineHandler	1	1	1	0
jgaapGUI	0.019	0.03	0.02	1
jigl.image.ops.levelOps.Clip	0.41	0.254	1	0
jipa.Main	0.001	0.007	0.2	0.002
jipa.Variable	0.174	0.174	1	0.573
lotus.core.Game	1	1	1	1
lotus.core.phases.Phase	1	1	1	1
macaw.presentation.Layer.MacawWorkBench	0.342	0.035	0.009	1
macaw.util.DisplayableListPanel	0.234	0.22	1	0.088
Menu	1	1	1	1
mindbright.ssh.SSHSCPGUIThread	0	0.078	0.008	0
mygrid.web._AvailableJobsResponse	1	1	1	0
mygrid.web._Fail	1	1	1	0
net.kencochrane.a4j.DAO.Product	0.605	0.174	0.162	0.654
net.kencochrane.a4j.file.FileUtil	0.101	0.977	0.595	0.767
net.sf.gaeappmanager.google.appengine.QuotaDetailsParser	1	1	1	1
net.sf.lavalamp.device.TimeOfDay	1	1	1	1
net.sf.sugar.fspath.cli.FSPPathExplorer	0.222	0.084	0.015	0.563
net.sf.sugar.fspath.FSPPathResult	1	1	1	1
net.sf.xbus.base.core.MessageHandler	1	0.964	1	0.833
net.sf.xbus.base.core.reflection.XBUSClassLoader	1	1	1	1
net.sf.xisemele.impl.OperationsHelperImpl	1	0.342	1	0
net.sourceforge.beanbin.reflect.MethodReflectionCriteria	0.001	0.005	0.002	1
net.sourceforge.beanbin.reflect.ReflectionSearch	1	1	1	0.08
net.sourceforge.ext4j.log.log4j.ExtrasPatternParser	1	1	1	1
net.sourceforge.ext4j.taglib.bo.Request	0.256	0.544	0.989	0
net.sourceforge.schemaspy.model.Table	0.153	0.497	0.168	0.346
net.sourceforge.schemaspy.model.xml.TableMeta	0.218	0.07	0.978	0.341
net.sourceforge.squirrel_sql.client.preferences.GlobalPreferencesSheet	1	1	1	1
net.sourceforge.squirrel_sql.fw.gui.CascadeInternalFramePositioner	0.002	0	0	0
net.virtualinfinity.atrobots.gui.renderers.RobotRenderer	0.016	0.035	0.003	0.81
net.virtualinfinity.atrobots.robot.Robot	0	0.01	0.001	0.81
nu.staldal.lagoon.LagoonCLI	0	0	0.002	0.019
nu.staldal.lagoon.LagoonGUI	1	1	1	1
org.apache.lucene.search.exposed.compare.NamedCollatorComparator	0.012	0.011	0.282	0.001
org.apache.lucene.search.exposed.ExposedTimSort	0.052	0.61	0.252	0.614
org.asphodel.DefaultRepositoryManager	1	1	1	1
org.character.parse.FromClient	1	1	1	1
org.databene.jdbcl.DBUtil	0	0	0	0
org.databene.jdbcl.SQLUtil	0	0	0	0
org.dom4j.io.DOMWriter	1	1	1	1
org.dom4j.tree.AbstractElement	0	0	0	0
org.exolab.jms.net.connector.DefaultConnectionPool	0.175	0.259	0.489	0.59
org.exolab.jms.net.socket.SocketRequestInfo	1	1	1	0
org.firebirdsql.encodings.EncodingFactory	0	0	0	0
org.firebirdsql.gds.impl.wire.AbstractJavaGDSImpl	1	1	1	1
org.firebirdsql.jdbc.FBProcedureCall	0.48	0	1	0
org.fixsuite.message.Library	1	0.37	0.356	0.342
org.fixsuite.message.view.TreeView	0.076	0.099	0.025	0.308
org.javathena.login.Login	0.001	0.001	0.032	0.285
org.javathena.login.UserManagement	0	0.009	0.001	0.548
org.javathena.utiles.ConfigurationManagement	0.122	0.521	0.731	0.067
org.jsecurity.util.AntPathMatcher	0	0	0.001	0.989
org.jsecurity.web.DefaultWebSecurityManager	0.891	0.896	0.945	0
org.jsecurity.web.filter.authc.BasicHttpAuthenticationFilter	1	1	1	1
org.mozilla.javascript.Parser	0	0	0	0

Tablo EK-1.1. Her bir sınıfın 20 bağımsız koşma üzerinden Dal Kapsamı metriğine göre Wilcoxon test sonuçları.

Sınıf Adı	MABC/ABC	MABC/MGA	MABC/SGA	MABC/RS
org.objectweb.asm.jip.ClassReader	0.561	0.747	0.871	0.884
org.objectweb.asm.jip.commons.LocalVariablesSorter	0.818	0.128	0.745	0
org.objectweb.asm.jip.MethodWriter	0.01	0.013	0.113	0.714
org.pdfsam.guiclient.business.thumbnails.callables.JPodThumbnailCallable	1	1	1	1
org.petsoar.order.CreditCardInfo	1	1	1	1
org.petsoar.search.lucene.DefaultLuceneDocumentFactory	1	1	1	1
org.saxpath.Axis	1	1	1	1
org.sourceforge.ifx.basetypes.IFXObject	0.342	1	1	1
org.templateit.Region	1	1	1	0
org.templateit.WorkbookParser	0.004	0	0	0.112
osa.ora.server.admin.ui.UpdateUserPanel	1	1	1	1
osa.ora.server.client.ui.utils.FontChooserDialog	1	1	1	1
Scanner	0.933	0.038	0	0
server.ServerGameModel	0.943	0.754	0.943	0.321
simulator.CA.ConsumerAgent	0.637	0.39	0.298	0.62
teder.TederFrame	1	1	1	1
uk.me.jockmacmad.jstate.state.Transition	1	1	0.317	0
umd.cs.shop.JSshop	0.001	0.014	0.454	0.018
umd.cs.shop.JSPredicateForm	0.053	0.966	0.327	0
umd.cs.shop.JSState	0.151	0.615	0.153	0.863
umd.cs.shop.JSTerm	0	0	0	0
visu.handball.moves.actions.CreateMovePdfAction	1	1	1	1
visu.handball.moves.xml.HandballModelReader	1	1	1	0
weka.classifiers.bayes.net.search.ci.ICSSearchAlgorithm	0.049	0	0.587	0.602
weka.classifiers.Evaluation	0.068	0.003	0.256	0.113
weka.core.FindWithCapabilities	0	0	0	0.008
weka.core.Memory	0.019	0.011	0	0.161
wheel.asm.ClassReader	0.267	0.726	0.766	0.882
wheel.asm.ClassWriter	0.765	0.553	0.704	0
wheel.asm.FieldWriter	0	0	0	0
wheel.util.DynamicSelectModel	0.433	0.055	0.131	0.01

Tablo EK-1.2. Her bir sınıf için bağımsız 20 koşmanın ortalama Dal Kapsamı değerleri.

Sınıf Adı	MABC	ABC	MGA	SGA	RS
accessories.plugins.time.JDayChooser	0.751	0.73	0.736	0.732	0.759
apbs_mem_gui.Main	0.003	0.003	0.003	0.003	0.003
bcry.battlecryGUI	0.026	0.026	0.026	0.026	0.025
bcry.bcGenerator	0.15	0.129	0.09	0.092	0.285
bible.obj.Verse	0.594	0.594	0.594	0.594	0.551
bible.util.Queue	1	1	1	1	0.98
bierse.controller.DefaultSettingsController	0.326	0.333	0.333	0.333	0.315
bierse.model.Drink	0.992	0.989	0.991	0.99	0.969
Capture	0.103	0.103	0.103	0.103	0.103
ch.bfh.egov.nutzenportfolio.common.AuswertungGrafik	0.279	0.333	0.333	0.333	0.278
ch.bfh.egov.nutzenportfolio.service.fragebogen.NaOpNuDaoService	0.104	0.104	0.104	0.104	0.104
client.network.ForwardingObserver	0.735	0.919	0.83	0.823	0.607
com.aeltis.azureus.ui.swt.feature.FeatureManagerUIListener	0.025	0.025	0.025	0.024	0.025
com.aeltis.azureus.ui.swt.skin.SWTSkinObjectContainer	0.017	0.017	0.017	0.017	0.005
com.allenstudio.ir.ui.MainMenu	0.741	0.741	0.741	0.741	0.741
com.allenstudio.ir.util.XmlElement	0.941	0.932	0.946	0.939	0.829
com.atlassw.tools.eclipse.checkstyle.projectconfig.ProjectConfiguration	0.108	0.108	0.108	0.108	0.108
com.browsersoft.aacs.User	0.994	0.991	0.993	0.994	0.894
com.browsersoft.openhre.hl7.impl.regular.ExpressionImpl	1	1	1	1	0.989
com.densebrain.rif.server.transport.WebServiceDescriptor	1	1	1	1	0.91
com.eteks.sweethome3d.j3d.Room3D	0.003	0.003	0.003	0.003	0.002
com.eteks.sweethome3d.SweetHome3D	0.164	0.156	0.156	0.157	0.153
com.eteks.sweethome3d.viewcontroller.HomeController	0.102	0.089	0.081	0.093	0.114
com.eteks.sweethome3d.viewcontroller.RoomController	0.039	0.042	0.046	0.039	0.039
com.gbshape.dbe.sapdb.SapdbTableList	0.308	0.308	0.308	0.308	0.308
com.gbshape.dbe.utils.DBHelper	0.441	0.387	0.384	0.412	0.496
com.gotoservers.ftp.FtpApplet	0.034	0.034	0.034	0.034	0.034
com.hf.sfm.crypt.Base64	1	1	1	1	0.991
com.hf.sfm.util.Loader	0.438	0.4	0.415	0.405	0.449
com.ib.client.ComboLeg	1	0.997	1	1	0.857
com.ib.client.ExecutionFilter	0.988	0.998	0.997	0.992	0.706
com.imsmart.servlet.HTMLFilter	1	1	1	1	1
com.liferay.portlet.documentlibrary.model.DLSyncWrapper	1	1	1	1	0.979
com.lts.event.SimpleKeyListenerHelper	1	1	1	1	0.856
com.lts.io.DirectoryScanner	0.774	0.702	0.726	0.715	0.848
com.lts.swing.combobox.SimpleComboBox	0.879	0.856	0.855	0.852	0.897
com.mentorgen.tools.profile.runtime.Profile	0.858	0.885	0.87	0.852	0.825
com.pmdesigns.jvc.tools.JVCPParser	0.471	0.444	0.442	0.455	0.449
com.pmdesigns.jvc.tools.JVCPParserTokenManager	0.169	0.144	0.151	0.154	0.163
com.pmdesigns.jvc.tools.TokenMgrError	0.931	0.986	0.97	0.988	0.809
com.sap.managementconsole.soap.axis.sapcontrol.HeapInfo	0.929	0.932	0.93	0.931	0.689
com.sap.managementconsole.soap.axis.sapcontrol.J2EEApplicationAlias	0.926	0.927	0.928	0.928	0.723
com.soops.CEN4010.JMCA.JMCAAnalyzer	0.125	0.145	0.096	0.108	0.121
com.soops.CEN4010.JMCA.JParser.JavaCharStream	0.771	0.691	0.693	0.706	0.643
com.soops.CEN4010.JMCA.JParser.JavaParser	0	0	0	0	
com.soops.CEN4010.JMCA.JParser.JavaParserTokenManager	0.435	0.327	0.364	0.362	0.453
com.werken.saxpath.XPathLexer	0.826	0.69	0.703	0.72	0.683
com.yahoo.platform.yui.compressor.JavaScriptCompressor	0.236	0.237	0.24	0.242	0.299

Tablo EK-1.2. Her bir sınıf için bağımsız 20 koşmanın ortalama Dal Kapsamı değerleri.

Sınıf Adı	MABC	ABC	MGA	SGA	RS
com.yahoo.platform.yui.compressor.ScriptOrFnScope	0.769	0.769	0.769	0.769	0.759
com.yahoo.platform.yui.compressor.YUICompressor	0.36	0.355	0.356	0.357	0.376
Convert	0.113	0.114	0.113	0.11	0.115
corina.graph.GrapherPanel	0.003	0.003	0.003	0.003	0.003
corina.graph.StandardPlot	0.087	0.087	0.087	0.087	0.087
corina.site.SiteDB	0.182	0.192	0.172	0.175	0.183
de.beiri22.stringincrementor.helper.IndexedString	0.974	0.982	0.976	0.97	0.955
de.beiri22.stringincrementor.StringIncrementor	0.837	0.849	0.851	0.854	0.814
de.huxhorn.lilith.data.access.AccessEvent	0.932	0.952	0.948	0.943	0.659
de.huxhorn.lilith.swing.preferences.ConditionTableModel	0.862	0.862	0.862	0.86	0.818
de.huxhorn.lilith.swing.table.renderer.MethodRenderer	0.806	0.81	0.803	0.81	0.77
de.huxhorn.lilith.swing.ViewActions	0.008	0.008	0.008	0.008	0.007
de.outstare.fortbattleplayer.model.impl.SimpleSector	0.851	0.865	0.876	0.865	0.839
de.outstare.fortbattleplayer.statistics.BattleStatistics	0.751	0.742	0.732	0.697	0.697
de.paragon.explorer.event.ExplorerFrameEventConverter	0.017	0.017	0.017	0.017	0.017
de.paragon.explorer.model.AttributeModelComparator	0.868	0.842	0.935	0.906	0.776
de.progra.charting.CoordSystemUtilities	0.55	0.562	0.527	0.541	0.528
de.progra.charting.DefaultChart	0.392	0.3	0.334	0.332	0.365
dsachat.client.gui.InternalChatFrame	0.043	0.043	0.043	0.043	0.043
dsachat.server.Handler	0.431	0.391	0.413	0.396	0.454
edu.uiuc.ndiipp.hubandspoke.profile.HaSMETSWebValidator	0.01	0.01	0.01	0.01	0.01
edu.uiuc.ndiipp.hubandspoke.profile.TestHaSMETSProfile	0.011	0.011	0.011	0.011	0.011
fi.vtt.noen.mfw.bundle.probe.shared.ProbeInformation	0.86	0.937	0.941	0.881	0.687
fi.vtt.noen.testgen.observations.formatter.DaikonFormatter	0.697	0.696	0.658	0.674	0.682
fps370.Fps370Panel	0.007	0.007	0.007	0.007	0.007
fr.pingtimeout.jtail.gui.action.IndexFileAction	0.308	0.308	0.308	0.308	0.296
fr.pingtimeout.jtail.gui.view.JTailPanel	0.804	0.802	0.741	0.239	0.652
framework.util.jdbc.JDBCUtils	0.339	0.345	0.347	0.346	0.303
framework.util.StringUtils	0.937	0.914	0.927	0.925	0.801
freemind.extensions.ExportHook	0.318	0.326	0.317	0.317	0.278
geo.google.datamodel.GeoStatusCode	1	1	1	1	1
geo.google.mapping.AddressToUsAddressFunctor	0.623	0.7	0.702	0.677	0.506
ghm.follow.gui.TabbedPane	0.375	0.36	0.334	0.355	0.352
ghm.follow.search.SearchableTextPane	0.857	0.806	0.843	0.736	0.726
glengineer.agents.GroupAgent	0.613	0.65	0.658	0.657	0.658
glengineer.blocks.Block	0.957	0.957	0.957	0.957	0.789
gui.AudioManager	0.469	0.469	0.469	0.469	0.425
gui.GroupPanel	0.04	0.04	0.04	0.04	0.04
httpanalyzer.HttpAnalyzerView	0.015	0.015	0.015	0.015	0.015
httpanalyzer.Password	0.804	0.804	0.804	0.804	0.804
ioproject.server.network.ClientGroup	0.855	0.925	0.922	0.894	0.783
ipac.BinaryCalculate	0.905	0.867	0.892	0.872	0.928
ipac.WhoIS	0.515	0.452	0.451	0.437	0.475
jahuwaldt.plot.PlotAxis	0.997	0.999	0.995	0.999	0.932
jaw.entrada.Abrir	0.192	0.192	0.192	0.192	0.192
jcmdline.HelpCmdLineHandler	0.634	0.634	0.634	0.634	0.616
jgaapGUI	0.826	0.815	0.816	0.813	0.826
jigl.image.ops.levelOps.Clip	0.941	0.942	0.947	0.932	0.862
jipa.Main	0.606	0.552	0.568	0.583	0.65
jipa.Variable	0.996	1	1	0.996	0.998
lotus.core.Game	0.417	0.417	0.417	0.417	0.417
lotus.core.phases.Phase	0.5	0.5	0.5	0.5	0.5
macaw.presentation.Layer.MacawWorkBench	0.174	0.172	0.165	0.161	0.174
macaw.util.DisplayableListPanel	0.256	0.367	0.355	0.262	0.146
Menu	0.06	0.06	0.06	0.06	0.06
mindbright.ssh.SSHSCPGUIThread	0.439	0.372	0.41	0.4	0.549
mygrid.web._AvailableJobsResponse	0.821	0.821	0.821	0.821	0.79
mygrid.web._Fail	0.821	0.821	0.821	0.821	0.788
net.kencochrane.a4j.DAO.Product	0.094	0.095	0.097	0.097	0.092
net.kencochrane.a4j.file.FileUtil	0.455	0.476	0.458	0.463	0.451
net.sf.gaeappmanager.google.appengine.QuotaDetailsParser	0.149	0.149	0.149	0.149	0.149
net.sf.lavalamp.device.TimeOfDay	1	1	1	1	1
net.sf.sugar.fspath.cli.FSPPathExplorer	0.682	0.665	0.656	0.651	0.696
net.sf.sugar.fspath.FSPPathResult	1	1	1	1	1
net.sf.xbus.base.core.MessageHandler	0.176	0.176	0.177	0.176	0.172
net.sf.xbus.base.core.reflection.XBUSClassLoader	0.176	0.176	0.176	0.176	0.176
net.sf.xisemele.impl.OperationsHelperImpl	0.556	0.556	0.557	0.556	0.507
net.sourceforge.beanbin.reflect.MethodReflectionCriteria	0.83	0.755	0.77	0.783	0.83
net.sourceforge.beanbin.reflect.ReflectionSearch	0.659	0.659	0.659	0.659	0.655
net.sourceforge.ext4j.log.log4j.ExtrasPatternParser	1	1	1	1	1
net.sourceforge.ext4j.taglib.bo.Request	0.926	0.92	0.928	0.926	0.784
net.sourceforge.schemaspy.model.Table	0.036	0.035	0.036	0.034	0.034
net.sourceforge.schemaspy.model.xml.TableMeta	0.368	0.529	0.598	0.397	0.256
net.sourceforge.squirrel_sql.client.preferences.GlobalPreferencesSheet	0.078	0.078	0.078	0.078	0.078
net.sourceforge.squirrel_sql_fw.gui.CascadeInternalFramePositioner	0.825	0.898	0.952	0.912	0.749
net.virtualinfinity.atrobots.gui.renderers.RobotRenderer	0.646	0.508	0.528	0.477	0.663
net.virtualinfinity.atrobots.robot.Robot	0.587	0.574	0.578	0.576	0.59
nu.staldal.lagoon.LagoonCLI	0.302	0.275	0.282	0.282	0.312
nu.staldal.lagoon.LagoonGUI	0.016	0.016	0.016	0.016	0.016
org.apache.lucene.search.exposed.compare.NamedCollatorComparator	0.802	0.848	0.849	0.832	0.715
org.apache.lucene.search.exposed.ExposedTimSort	0.15	0.235	0.17	0.172	0.133
org.asphodel.DefaultRepositoryManager	0.024	0.024	0.024	0.024	0.024
org.character.parse.FromClient	1	1	1	1	1
org.databene.jdbcl.DBUtil	0.566	0.494	0.526	0.525	0.618
org.databene.jdbcl.SQLUtil	0.882	0.813	0.822	0.82	0.819
org.dom4j.io.DOMWriter	0.029	0.029	0.029	0.029	0.029
org.dom4j.tree.AbstractElement	0.774	0.622	0.626	0.657	0.825
org.exolab.jms.net.connector.DefaultConnectionPool	0.285	0.287	0.289	0.288	0.287

Tablo EK-1.2. Her bir sınıf için bağımsız 20 koşmanın ortalama Dal Kapsamı değerleri.

Sınıf Adı	MABC	ABC	MGA	SGA	RS
org.exolab.jms.net.socket.SocketRequestInfo	1	1	1	1	0.872
org.firebirdsql.encodings.EncodingFactory	0.963	0.925	0.933	0.932	0.959
org.firebirdsql.gds.impl.wire.AbstractJavaGDSImpl	0	0	0	0	
org.firebirdsql.jdbc.FBProcedureCall	0.854	0.843	0.889	0.854	0.812
org.fixsuite.message.Library	0.237	0.243	0.229	0.229	0.229
org.fixsuite.message.view.TreeView	0.311	0.281	0.283	0.272	0.317
org.javathena.login.Login	0.427	0.385	0.389	0.405	0.448
org.javathena.login.UserManagement	0.164	0.154	0.157	0.156	0.163
org.javathena.utiles.ConfigurationManagement	0.472	0.479	0.508	0.489	0.476
org.jsecurity.util.AntPathMatcher	0.729	0.685	0.697	0.701	0.73
org.jsecurity.web.DefaultWebSecurityManager	0.678	0.673	0.678	0.681	0.609
org.jsecurity.web.filter.authc.BasicHttpAuthenticationFilter	0.444	0.444	0.444	0.444	0.444
org.mozilla.javascript.Parser	0.294	0.257	0.256	0.265	0.351
org.objectweb.asm.jip.ClassReader	0.281	0.256	0.296	0.28	0.287
org.objectweb.asm.jip.commons.LocalVariablesSorter	0.749	0.741	0.792	0.741	0.601
org.objectweb.asm.jip.MethodWriter	0.239	0.21	0.206	0.224	0.242
org.pdfsam.guiclient.business.thumbnails.callables.JPodThumbnailCallable	0.158	0.158	0.158	0.158	0.158
org.petsoar.order.CreditCardInfo	1	1	1	1	1
org.petsoar.search.lucene.DefaultLuceneDocumentFactory	0.511	0.511	0.511	0.511	0.511
org.saxpath.Axis	1	1	1	1	1
org.sourceforge.ifx.basetypes.IFXObject	0.431	0.426	0.431	0.431	0.431
org.templateit.Region	1	1	1	1	0.944
org.templateit.WorkbookParser	0.225	0.269	0.269	0.312	0.213
osa.ora.server.admin.ui.UpdateUserPanel	0	0	0	0	0
osa.ora.server.client.ui.utils.FontChooserDialog	0.16	0.16	0.16	0.16	0.16
Scanner	0.109	0.107	0.117	0.02	0.039
server.ServerGameModel	0.07	0.071	0.074	0.071	0.071
simulator.CA.ConsumerAgent	0.118	0.114	0.119	0.118	0.12
teder.TederFrame	0.014	0.014	0.014	0.014	0.014
uk.me.jockmacmad.jstate.state.Transition	1	1	1	0.998	0.902
umd.cs.shop.JSJshop	0.26	0.389	0.346	0.314	0.26
umd.cs.shop.JSPredicateForm	0.719	0.676	0.71	0.687	0.783
umd.cs.shop.JSSState	0.506	0.488	0.52	0.487	0.515
umd.cs.shop.JSTerm	0.684	0.609	0.617	0.624	0.728
visu.handball.moves.actions.CreateMovePdfAction	0.038	0.038	0.038	0.038	0.038
visu.handball.moves.xml.HandballModelReader	0.355	0.355	0.355	0.355	0.336
weka.classifiers.bayes.net.search.ci.ICSSearchAlgorithm	0.22	0.174	0.128	0.203	0.248
weka.classifiers.Evaluation	0.29	0.248	0.228	0.26	0.311
weka.core.FindWithCapabilities	0.801	0.706	0.748	0.697	0.82
weka.core.Memory	0.777	0.764	0.763	0.759	0.769
wheel.asm.ClassReader	0.333	0.321	0.32	0.33	0.351
wheel.asm.ClassWriter	0.897	0.899	0.906	0.905	0.812
wheel.asm.FieldWriter	0.967	0.997	1	0.998	0.841
wheel.util.DynamicSelectModel	0.732	0.712	0.689	0.695	0.82
1. Çeyrek	0.167	0.169	0.159	0.161	0.163
Ortanca	0.503	0.491	0.509	0.473	0.503
3. Çeyrek	0.854	0.851	0.857	0.852	0.792
Ortalama	0.516	0.511	0.513	0.505	0.491
Sıralamaların Ortalaması	2.669	3.067	2.797	3.092	3.375

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı, Soyadı : Ömür ŞAHİN
Uyruğu : Türkiye (T.C.)
Doğum Tarihi ve Yeri : 01.01.1991 Kocasinan/KAYSERİ
Telefon : 0 352 207 66 66
Belgegeçer : 0 352 437 57 84
E-posta : omur@erciyes.edu.tr
Adres : Erciyes Üniversitesi Mühendislik Fakültesi
 Bilgisayar Mühendisliği Bölümü
 38039, Melikgazi KAYSERİ TÜRKİYE

EĞİTİM

Derece	Kurum	Mez.Yılı
Lise	75. Yıl Cumhuriyet Anadolu Lisesi, KAYSERİ	2009
Lisans	Erciyes Üniversitesi Bilgisayar Müh., KAYSERİ	2014
Yüksek Lisans	Erciyes Üni. Fen Bilimleri Ens. Bilgisayar Müh. ABD, KAYSERİ	2016
Doktora	Erciyes Üni. Fen Bilimleri Ens. Bilgisayar Müh. ABD, KAYSERİ	2020

İŞ DENEYİMİ

Yıl	Kurum	Görev
2014-Halen	ERÜ Mühendislik Fakültesi Bilgisayar Müh. Bölümü, KAYSERİ	Araştırma Görevlisi

YAYINLAR

SCI, SSCI, AHCI İndekslerine Giren Dergilerde Yayımlanan Makaleler

O. Sahin, B. Akay, Comparisons of metaheuristic algorithms and fitness functions on software test data generation, Applied Soft Computing, ss.1202-1204, 2016

O. Sahin, B. Akay, D. Karaboga, Archive-Based Multi-Criteria Artificial Bee Colony Algorithm for Whole Test Suite Generation, Engineering Science and Technology, an International Journal, 2020.

TR Dizinli Dergilerde Yayımlanan Makaleler

O. Sahin, B. Akay, Yinelemeli F-Yarış Algoritması ile Yapay Arı Koloni Algoritmasının Kontrol Parametrelerinin Ayarlanması, Erciyes Üniversitesi Fen Bilimleri Enstitüsü Fen Bilimleri, 2020

Uluslararası Katılımlı Konferans Yayınları

B. Akay, O. Sahin, Döngü İçeren Programlar İçin Test Verisi Üretiminde Meta-sezgisel Algoritmaların Kontrol Parametresi Analizi, International Conference on Computer Science and Engineering, ss.306-313, Tekirdağ, Türkiye, 2016

O. Sahin, B. Akay, Döngü İçeren Kod Parçaları için Test Verisi Üretiminde Meta-sezgisel Algoritmaların İncelemesi, 1st International Conference on Engineering Technology and Applied Sciences, ss.362-369, Afyon, Türkiye, 2016

Ulusal Konferans Yayınları

O. Sahin, B. Akay, Öğrenme-Öğretme Temelli ve Armoni Arama Algoritmaları ile Test Verisi Üretimi, 10. Ulusal Yazılım Mühendisliği Sempozyumu, ss.154-165, Çanakkale, Türkiye, 2016

SCI Dizinli Dergi Sistemlerine Yüklenen Revizyondaki Çalışmalar

O. Sahin, B. Akay, A Discrete Dynamic Artificial Bee Colony with Hyper-Scout for RESTful Web Service API Test Suite Generation