T.C.
ERCİYES ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

YAZILIM TEST VERİSİ ÜRETİMİNDE YAPAY ZEKA TEKNİKLERİNİN PERFORMANS ANALİZİ

Hazırlayan
Ömür ŞAHİN

Danışman
Doç. Dr. Bahriye AKAY

Yüksek Lisans Tezi

Bu çalışma Erciyes Üniversitesi Bilimsel Araştırma Projeleri Birimi tarafından FYL-2016-6454 kodlu proje ile desteklenmiştir

Temmuz 2016
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

"Yazılım Test Verisi Üretiminde Yapay Zeka Tekniklerinin Performans Analizi" adlı Yüksek Lisans tezi, Erciyes Üniversitesi Lisansüstü Tez Önerisi ve Tez Yazma Yönergesi'ne uygun olarak hazırlanmıştır.

Tezi Hazırlayan
Ömür ŞAHİN

Danışman
Doç. Dr. Bahriye AKAY

Bilgisayar Mühendisliği ABD Başkanı
Prof. Dr. Derviş KARABOĞA

Doç. Dr. Bahriye AKAY danışmanlığında **Ömür ŞAHİN** tarafından hazırlanan **"Yazılım Test Verisi Üretiminde Yapay Zekâ Tekniklerinin Performans Analizi"** adlı bu çalışma, jürimiz tarafından Erciyes Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalında **Yüksek Lisans** tezi olarak kabul edilmiştir.

../../2016

JÜRİ :

Danışman : Doç. Dr. Bahriye AKAY

Üye : Prof. Dr. Oya KALIPSIZ

Üye : Prof. Dr. Derviş KARABOĞA

ONAY :

Bu tezin kabülü Enstitü Yönetim Kurulunun 02/08/2016 tarih ve 2016/34-03 sayılı kararı ile onaylanmıştır.

02/08/2016

Prof. Dr. Mehmet AKKURT
Enstitü Müdürü

ÖNSÖZ / TEŞEKKÜR

Tez çalışmam boyunca aktardığı bilgi ve tecrübeleri ile yol gösteren, çalışma disiplini ve sahip olduğu bilgi birikimi ile örnek aldığım danışmanım Doç. Dr. Bahriye AKAY'a, tecrübe ve yardımlarını esirgemeyen değerli hocam Bilgisayar Mühendisliği Bölüm Başkanı Prof. Dr. Derviş KARABOĞA'ya, FYL-2016-6454 kodlu proje desteğinden dolayı Erciyes Üniversitesi Bilimsel Araştırma Projeleri Birimi'ne, hayat boyu bilgi ve tecrübeleri ile sürekli yol gösteren abilerim Necip YILDIZ ve İlhan YILDIZ'a, tez çalışmam boyunca yoldaşlık eden ve hiçbir anında yalnız bırakmayan Gizem AYTEKİN ve Aydan ÇAĞLAR'a ve en önemlisi her zaman yanımda olan aileme teşekkür ederim.

Bu tezi, maddi ve manevi hiçbir fedakârlıktan kaçınmayıp desteklerini hiçbir zaman esirgemeyen sevgili aileme ithaf ediyorum.

Ömür ŞAHİN

Kayseri, 2016

YAZILIM TEST VERİSİ ÜRETİMİNDE YAPAY ZEKA TEKNİKLERİNİN PERFORMANS ANALİZİ

Ömür ŞAHİN
Erciyes Üniversitesi, Fen Bilimleri Enstitüsü
Yüksek Lisans Tezi, Temmuz 2016
Danışman : Doç. Dr. Bahriye AKAY

ÖZET

Yazılımların zamanında teslim edilmemesi, bütçeyi aşması, eksik fonksiyonlarla teslim edilmesi gibi yazılım krizlerinin en büyük sebebi yazılımlarla ilgili testlerin en başından itibaren yapılmamasıdır. Hatalar ne kadar geç fark edilirse düzeltme maliyeti ve ayrıca projenin toplam maliyeti o denli artmaktadır. Doğru yapılacak test işlemleri sayesinde yanlışlar erken aşamalarda görülüp düzeltilebilir böylece zaman ve maliyetten tasarruf sağlanabilir. Test işlemlerinde kullanılan test verisi yapılan testin etkinliğini oldukça etkilemektedir. Bu çalışmada arama tabanlı test verisi üretiminde Yapay Arı Koloni, Diferansiyel Gelişim, Parçacık Sürü Optimizasyon ve Ateş böceği algoritmalarının temel problemler ve döngü içeren problemlerdeki arama yetenekleri kıyaslanmıştır. Temel problemler triangle, even-odd, quadratic equation, largest number, remainder, leap year kod parçacıklarından, döngü içeren problemler ise asal, armstrong ve palindrom sayılarını bulan kod parçacıklarından oluşmaktadır. Meta-sezgisellerin farklı parametre konfigürasyonları incelenmiş ve uygun kontrol parametreleri önerilmiştir. Standart sapma ve üretilen değerler analiz edilerek algoritmaların kontrol parametrelerine olan bağlılığı ortaya çıkarılmıştır. Algoritmalar için uygun parametreler belirlendikten sonra algoritmaların test verisi üretiminde performansları kıyaslanmıştır. Ayrıca rastgele arama ile de kıyaslamalar yapılmıştır. Tez çalışmasında incelenen bir değer konu da meta-sezgiselleri yönlendiren amaç fonksiyonlarının performans üzerindeki etkisidir. Yol tabanlı (path-based), benzerlik tabanlı (dissimalarity-based) ve yakınlaşmaya tabanlı (approximation level + branch distance) amaç fonksiyonları incelenmiştir.

Anahtar Kelimeler: Meta-sezgisel algoritmalar, yapay arı koloni algoritması (ABC), parçacık sürü optimizasyonu (PSO), diferansiyel gelişim algoritması (DE), ateş böceği algoritması (FA), test verisi üretimi.

PERFORMANCE ANALYSIS OF ARTIFICIAL INTELLIGENCE ON SOFTWARE TEST DATA GENERATION

Ömür ŞAHİN
Erciyes University, Graduate School of Natural and Applied Sciences
M.Sc. Thesis, July 2016
Supervisor: Doç. Dr. Bahriye AKAY

ABSTRACT

The main reason of software crisis such as delivering late, exceeding the project budget, delivering with missing functions is that the software testing is not carried out from the beginning of software production cycle. As the testing is performed at final stages, an increase in the expenses of the maintenance and then the total cost occurs. Accurate testing activities provide to find errors in early stages and pass on huge savings in total cost. Test data used in testing affects the efficiency of the process. In this study Artificial Bee Colony, Differential Evolution, Particle Swarm Optimization and Firefly algorithms are used to generate search based software test data generation. Basic problems consist of triangle, even-odd, quadratic equation, largest number, remainder, leap year code fragments while the other group consist of problems that contain loops in code fragments such as prime, armstrong and palindrome number problems. For meta-heuristics, different control parameter configurations are examined and appropriate values are recommended. Dependency of the algorithms on the control parameters are analysed based on standard deviation and the coverage performance. Algorithms are using the best parameter configurations and their performances are compared using mean, median and standard deviation metrics. Moreover, the meta-heuristics are compared to Random Search. The objective functions that guide the meta-heuristics are also investigated. Path-based, dissimilarity-based and approximation level + branch distance based fitness functions are considered in this study.

Keywords: Meta-heuristic algorithms, artificial bee colony algorithm (ABC), particle swarm optimization (PSO), differential evolution (DE), firefly algorithm (FA), test data generation.

İÇİNDEKİLER

YAZILIM TEST VERİSİ ÜRETİMİNDE
YAPAY ZEKA TEKNİKLERİNİN PERFORMANS ANALİZİ

BİLİMSEL ETİĞE UYGUNLUK SAYFASI i

YÖNERGEYE UYGUNLUK SAYFASI ii

KABUL VE ONAY iii

ÖNSÖZ / TEŞEKKÜR iv

ÖZET v

ABSTRACT vi

İÇİNDEKİLER vii

TABLOLAR LİSTESİ x

ŞEKİLLER LİSTESİ xii

KISALTMALAR xiii

GİRİŞ **1**

1. BÖLÜM
YAZILIM MÜHENDİSLİĞİ VE TEST

1.1. Yazılım Mühendisliği **7**

1.1.1. Yazılım Mühendisliği Nedir? **7**

1.1.2. Yazılım Geliştirme Fazları **8**

1.1.2.1. Tanımlama **8**

1.1.2.2. Ayrıntılandırma **8**

1.1.2.3. Gerçekleştirme **8**

1.1.2.4. Değerlendirme **9**

1.1.2.5. Geçiş **9**

1.1.2.6. Yönetim **9**

1.1.3. Yazılım Gereksinim Belirtimleri **9**

1.1.4. Yazılım Süreç Modellerinde Yazılım Testleri 11
1.2. Yazılım Testi 11
1.2.1. Yazılım Testi Nedir? 11
1.2.2. Yazılım Testinin Önemi 12
1.3. Yazılım Test Düzeyleri 14
1.3.1. Birim Testler 15
1.3.2. Entegrasyon Testi 15
1.3.3. Sistem Testleri 16
1.3.4. Kabul Testleri 17
1.4. Yazılım Test Teknikleri 17
1.4.1. Kara Kutu Testi 17
1.4.2. Saydam Kutu Testi 18
1.4.3. Gri Kutu Testi 19
2. BÖLÜM
ARAMA TABANLI TEST VERİSİ ÜRETİMİ
2.1. Arama Tabanlı Test Verisi Üretimi 21
2.2. Popülasyon Tabanlı Meta-sezgisel Algoritmalar 24
2.2.1. Parçacık Sürü Optimizasyonu 25
2.2.2. Yapay Arı Koloni Algoritması 26
2.2.3. Diferansiyel Gelişim Algoritması 27
2.2.4. Ateş Böceği Algoritması 29
3. BÖLÜM
META-SEZGİSEL ALGORİTMALARIN TEST VERİSİ ÜRETİMİNDE KULLANILMASI
3.1. Kullanılan Platform ve Mimariler 32
3.2. Temel Problemler Üzerinde Analizler 34
3.2.1. Parametre Analizi 36
3.2.2. Meta-sezgisel Algoritmaların Kıyaslanması 38
3.2.2.1. Yol Tabanlı Amaç Fonksiyonu Kıyaslaması 38
3.2.2.2. Benzerlik Tabanlı Amaç Fonksiyonu Kıyaslaması 42

3.2.2.3. Approximation Level+Branch Distance Tabanlı Amaç Fonksiyonu Kıyaslaması . . . 42

3.2.3. Amaç Fonksiyonu Analizi . . . 43

3.2.4. Rastgele Arama ile Kıyaslama . . . 43

3.3. Döngü İçeren Problemler . . . 44

3.3.1. Parametre Analizi . . . 45

3.3.2. Meta-sezgisel Algoritmaların Kıyaslanması . . . 48

3.3.2.1. Yol Tabanlı Amaç Fonksiyonu Kıyaslaması . . . 48

3.3.2.2. Approximation Level + Branch Distance Tabanlı Amaç Fonksiyonu Kıyaslaması . . . 50

3.3.3. Amaç Fonksiyonu Analizi . . . 53

4. BÖLÜM
TARTIŞMA, SONUÇ ve ÖNERİLER

4.1. Tartışma, Sonuç ve Öneriler . . . 55

KAYNAKLAR . . . 58

ÖZGEÇMİŞ . . . 65

TABLOLAR LİSTESİ

Tablo 3.1. Tracey'nin önermiş olduğu branch distance fonksiyonu. 33

Tablo 3.2. Temel problemlerin karakteristikleri. 35

Tablo 3.3. Test edilen kontrol parametreleri, NP yiyecek kaynağı sayısı ve D problemin boyutunu belirtmektedir. 36

Tablo 3.4. Temel problemlerde kullanılan her bir algoritma için önerilen kontrol parametreleri. 37

Tablo 3.5. Temel problemler için farklı ayarlarda koşturulan algoritmaların istatistiksel sonuçları. Ortalama $\mp$ Standart Sapma and Medyan değerleri verilmiştir. 37

Tablo 3.6. Temel problemler için algoritmaların yol tabanlı amaç fonksiyonu kullanarak elde ettikleri sonuçlar ortalama $\mp$ standart sapma olarak verilmiştir. Parantez içinde medyan değer, koyu renkte ise algoritmanın diğer algoritmalara kıyasla sırası verilmektedir. BS başarı sırasını ifade etmektedir. 39

Tablo 3.7. Temel problemler için algoritmaların dissimilarity tabanlı amaç fonksiyonu kullanarak elde ettiği sonuçlar ortalama $\mp$ standart sapma olarak verilmiştir. Parantez içinde medyan değer, koyu renkte ise algoritmanın diğer algoritmalara kıyasla sırası verilmektedir. BS başarı sırasını ifade etmektedir. 40

Tablo 3.8. Temel problemler için algoritmaların approximation level $+$ branch distance amaç fonksiyonu kullanarak elde ettiği sonuçlar ortalama $\mp$ standart sapma olarak verilmiştir. Parantez içinde medyan değer, koyu renkte ise algoritmanın diğer algoritmalara kıyasla sırası verilmektedir. BS başarı sırasını ifade etmektedir. 41

Tablo 3.9. Temel problemler için amaç fonksiyonlarının rank değerleri 44

Tablo 3.10. Temel problemler için RS algoritmasının sonuçları. 45

Tablo 3.11. Döngü içeren problemlerin karakteristikleri. 45

Tablo 3.12. Test edilen kontrol parametreleri, NP yiyecek kaynağı sayısı ve D problemin boyutunu belirtmektedir. 46

Tablo 3.13. Döngü içeren problemlerde kullanılan her bir algoritma için önerilen kontrol parametreleri. 46

Tablo 3.14. Döngü içeren problemler için farklı ayarlarda koşturulan algoritmaların istatistiksel sonuçları. Ortalama $\mp$ Standart Sapma and Medyan değerleri verilmiştir. 46

Tablo 3.15. Yol tabanlı amaç fonksiyonu ile döngü içeren problemler için elde edilen sonuçlar. 50

Tablo 3.16. Approximation level+branch distance tabanlı amaç fonksiyonu döngü içeren problemler için elde edilen sonuçlar. 51

Tablo 3.17. Döngü içeren problemler için amaç fonksiyonlarının rank değerleri. . 54

ŞEKİLLER LİSTESİ

Şekil G.1. Projelerin başarı oranı. 1

Şekil G.2. Hataların bulunma fazlarına göre düzeltilme maliyetleri. 3

Şekil 1.1. Gereksinim mühendisliği süreci. 10

Şekil 3.1. Çalışmada kullanılan test verisi üretimi mimarisi. 33

Şekil 3.2. Yol tabanlı amaç fonksiyonu için CFG örneği. 34

Şekil 3.3. Temel problemler için farklı ayarlarda koşturulan algoritmaların boxplot grafikleri. 37

Şekil 3.4. Asal sayılar kapsama miktarı. 47

Şekil 3.5. Armstrong sayıları kapsama miktarı. 48

Şekil 3.6. Palindrom sayılar kapsama miktarı. 49

Şekil 3.7. Asal sayılar problemi approximation level + branch distance amaç fonksiyonu ile elde edilen sonuçlar. 52

Şekil 3.8. Armstrong sayıları problemi approximation level + branch distance amaç fonksiyonu ile elde edilen sonuçlar. 52

Şekil 3.9. Palindrom sayılar problemi approximation level + branch distance amaç fonksiyonu ile elde edilen sonuçlar. 53

KISALTMALAR

ABC	:	Yapay arı koloni algoritması
PSO	:	Parçacık sürü optimizasyonu
ACO	:	Karınca koloni algoritması
DE	:	Diferansiyel gelişim algoritması
FA	:	Ateş böceği algoritması
RS	:	Rastgele arama
GA	:	Genetik algoritma

GİRİŞ

Bilgisayarların günlük hayatın parçası olması ile bilgisayar yazılımları da sistemlerin önemli unsurları olmuştur. Yazılımın bir mühendislik ürünü olmasıyla yazılım mühendisliği kavramı ortaya çıkmış ve yazılımın bir metodoloji takip edilerek üretilmesinin başarımı artıracağı düşünülmüştür. Bir araştırma şirketi olan Standish Group'un 2006'da yaptığı analizlere bakıldığında (Şekil G.1) başlanan projelerin yalnızca %35'inin istenen zamanda, anlaşılan bütçe ve talep edilen gereksinimleri karşılayacak şekilde tamamlandığı görülmektedir. Bu projelerin %19'u ise hiç tamamlanamamış ve iptal edilmiştir. %46'yı içeren pek çok proje ise geç teslim edilmiş, bütçeyi aşmış veya gereksinimleri sağlayamamıştır.



Şekil G.1. Projelerin başarı oranı. [1]

Yazılım mühendisliği, projenin zamanında uygun bütçe ve istenen gereksinimleri karşılayabilecek şekilde gerçekleşmesini amaçlayan disiplinler bütünüdür. Bunu yaparken projeyi bütün olarak ele almak yerine olabildiği kadar parçalara böler ve küçük parçalardaki sorunlar ile ilgilenip çözüm getirmeye çalışır. Yazılım yaşam döngüsü adı verilen bu yaklaşım ile proje çok daha kolay gerçekleştirilebilecek sıralı adımlar halini alır. Farklı adım sayılarına sahip çok sayıda yazılım yaşam döngüsü yaklaşımı sunulmuştur. Bunların en popülerlerinden biri şelale modelidir ve 1970'in sonlarına doğru literatüre sunulmuştur. Şelale modelinin pek çok çeşidi olmakla birlikte temel olarak

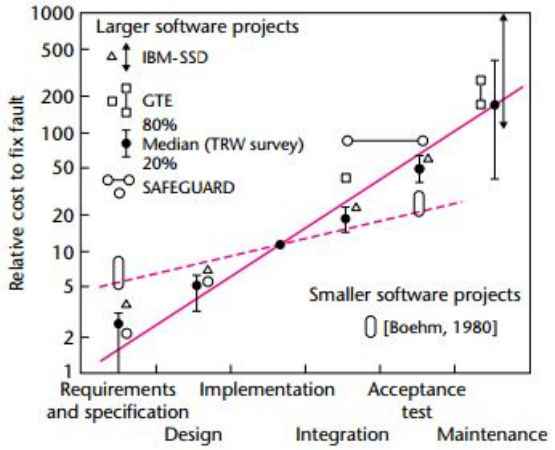
planlama, çözümleme, tasarım, geliştirme, teslimat sonrası bakım ve sonuçlandırma aşamalarından oluşmaktadır.

Planlama aşamasında proje için temel ihtiyaçlar belirlenmekte ve fizibilite çalışmaları yapılmaktadır. Çözümleme aşamasında sistemin gereksinimleri ortaya çıkarılır ve sonrasında tasarıma geçirilir. Tasarlama aşaması iki temel aşamadan meydana gelir. Öncelikle proje modüllere ayrılır (mimari tasarım). Daha sonra ise ayrılmış olan bu modüllerin detaylandırılması gerçekleştirilir (detaylı tasarım). Bu iki aşamanın ardından artık projenin nasıl gerçekleştirileceği belirlenmiş olur.

Tasarlanan projenin gerçekleştiriminde kodlama ve test aşamaları yer alır. Öncelikle birim testlerden geçen yazılım modülleri, her bir modülün tamamlanması sonunda entegrasyon testine girer ve modüllerin birbiri ile olan uyumu kontrol edilir. Daha sonra ise kullanıcı kabul testine tabi tutularak yazılımın son kullanıcı tarafından da ihtiyaçları karşılayıp karşılamadığı kararlaştırılır. Teslimat sonrası bakım ise daha önceden belirlenemeyen hataların düzeltilmesi (corrective) veya yazılıma istenen başka özellikleri eklenmesini içeren (adaptive, perfective) güncellemeler bütünüdür. Sonuçlandırma aşamasında ise yazılım artık gereksinimleri sağlayamamaktadır ve artık yazılıma destek verilemez [2].

Geliştirilen uygulamalarda çeşitli hataların meydana gelmesi oldukça olasıdır. Meydana gelen bu olası hataların neden olacağı sonuçlar ise uygulama türüne göre değişim göstermektedir. Uygulamalar müşteriye sunulmadan önce ürünün doğru çalışıyor olmasından ve kalitesinden emin olunmalıdır. Çünkü planlama, çözümleme ve tasarım aşamalarında proje taslak halinde olacağı için hataların giderilmesi çok daha az maliyetle gerçekleştirilebilmektedir. Hatalar ne kadar geç fark edilirse düzeltme maliyeti ve projenin toplam maliyeti de o denli artmaktadır (Şekil G.2) [2, 4]. Piyasaya sürülmüş bir uygulamanın tekrardan düzenleme ve geliştirme masrafları da oldukça büyük bir yük getirecektir. Doğru yapılacak test işlemleri sayesinde yanlışlar erken aşamalarda görülebilir, böylece zaman ve maliyetten tasarruf sağlanabilir. Uygulamanın doğru ve etkin bir performans ortaya koyması için mutlaka test edilmesi ayrıca bulunan hataların düzeltilmesi gerekmektedir. Bu yüzden geliştirilen uygulamalarda kullanılan bütçenin büyük bir çoğunluğu (%50'den fazla) uygulamanın test edilmesi için harcanmaktadır.

Bu yüksek maliyetli işlemi uygulamalar aracılığı ile gerçekleştirmek ve maliyeti düşürüp



Şekil G.2. Hataların bulunma fazlarına göre düzeltilme maliyetleri. [2]

uygulamanın ortaya koyulma süresini kısaltmak için pek çok araç geliştirilmiştir. Bu geliştirilen araçlar çeşitli yöntemler ile uygulamalar için test verileri üretmekte ve üretilen bu verileri çeşitli senaryolar ile denemektedir. Daha sonra elde ettiği sonuçları raporlayarak uygulama hakkında fikir elde etmeye yardımcı olmaktadır. Üretilen bu test verileri yeterli kalitede olmadığı zaman uygulamadaki hatalar görülemeyebilir. Tespit edilemeyen bu hatalar uygulamanın yanlış veya verimsiz çalışmasına sebep olabilir. Bu da uygulama alanına bağlı olarak çeşitli sorunların ortaya çıkmasına neden olabilir. Bu yüzden test verileri yeterli ve pek çok durumu kontrol edebilecek kalitede üretilmelidir.

Her ne kadar bu alanda çok fazla test aracı geliştirilmiş olsa da gerçek hayattaki uygulamalar oldukça karmaşık bir yapıda olduğu için etkin sonuçlar üretememektedirler. Bu yöntemlerin pek çoğunun oldukça fazla sorunu bulunmaktadır. Kaynak kod içerisindeki sınıflar, döngüler, işaretçiler gibi karmaşık yapıların kullanımı bu sorunların bazılarıdır. Bu yüzden bu alanda her geçen gün daha fazla çalışma yapılmakta ve yüksek başarım (kapsama'ya dayalı) elde etme, hız gibi konularda daha iyi sonuçlar elde edilmeye çalışılmaktadır.

Geçmişte yapılan çalışmalarda farklı etki alanlarına sahip uygulamaların test edilebilmesi için çeşitli teknikler sunulmuştur. Bu tekniklerden en çok kullanılanları;

- Sembolik çalıştırma (Symbolic Execution)
- Model tabanlı test (Model-based testing)
- Kombinasyonel test (Combinatorial testing)

- Rastgele testin bir türevi olan adaptif rastgele test (Adaptive 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 vardır.

Bu yöntemlerden sembolik çalıştırma olarak adlandırılan yöntem 1975 yılında King J. C. tarafından önerilmiştir. Giriş olarak somut veriler yerine sembolik veriler kullanılarak ve programın davranışı analiz edilir [5]. Programının iç yapısının bilinmesi gerektiği için saydam kutu test yöntemlerinden biridir. 70'li yıllarda ortaya atılmış olsa da o yıllarda sınırlamalı problemlerin çözümündeki olanaksızlık ve bilgisayarların yüksek hesaplama maliyetlerinin altından kalkmasındaki kısıtlar nedeniyle popülerleşememiştir. Bilgisayarların geçmişe oranla çok daha fazla güçlenmesi ve sınırlamalı problemlerin çözümündeki kat edilen aşamalar sembolik çalıştırmanın popüler olmasını sağlamış ve pek çok araştırma yapılmıştır. Bu araştırmaların temel amacı kapsama miktarını artırmak ve uygulamadaki hataları ortaya çıkarmak olmuştur [6–8].

Model tabanlı test yönteminde programın davranışı modellenmektedir. Model, girişe bağlı olarak bir çıkış üretmektedir. Kara kutu testini temel alır. Programdaki girişe bağlı olarak sonucun nasıl değiştiğini matematiksel veya aritmetik lojik olarak tanımlar.

Kombinasyonel test işleminde elimizdeki uzaydan örnek giriş parametreleri seçilir ve programın davranışı incelenir. Genelde programın ayarları kısmındaki çeşitli ayarlarda nasıl tepki vereceği belirlenmeye çalışılır.

Yakın etki alanındaki benzer test verileri büyük olasılıkla benzer hata değerine sahiptir. Bu temel düşünce ile üretilecek olan test verilerinin hataya sebebiyet vermesi veya vermemesi tahmin edilebilir. Adaptif rastgele test işlemi bu düşünce ile ortaya çıkmış test yöntemlerinden biridir ve rastgele test verisi üretme işlemini iyileştirmek için kullanılmaktadır [9, 10].

Arama tabanlı test verisi üretme, arama tabanlı uygulama geliştirmenin alanlarından biridir [11]. Bu alanda ilk yayın Webb Miller ve David Spooner tarafından 1976 yılında

çıkarılmıştır [12]. Bu yayın kayan noktalı sayıları (float) giriş alan programlar için o ana kadar geliştirilmemiş sembolik çalıştırma ve sınırlamalı problem çözümünü içeren bir teknik sunmaktadır [13]. Bu çalışmadan sonra 90'lı yıllara kadar çok fazla çalışma yapılmamıştır. Bu yıldan sonra Korel'in ortaya koyduğu çalışmalar [14, 15] ve 1992'de Xanthakis'in genetik algoritma ile bu konuda bir çözüme ulaşmış olması [16] ile arama tabanlı test verisi üretimi popüler hale gelmiştir.

Arama tabanlı test verisi üretimi meta-sezgisel algoritmalar ile gerçekleştirilmektedir. Meta-sezgisel algoritmaların test verilerini üretmesi bir amaç fonksiyonu yardımı ile sağlanır. Programın iç yapısının bilindiği ve verilerin buna göre üretildiği saydam kutu test tekniklerinden biri olan yapısal test [17–20] tekniği için çeşitli çalışmalar yapılmıştır. Programın iç yapısının bilinmediği, girişe bağlı olarak çıkış ile ilgilenen ve ilgili işlevi gerçekleştirip gerçekleştirmediğinin kontrolünün yapıldığı kara kutu test tekniklerinden biri olan fonksiyonel test [21] işlemi de oldukça popüler çalışma alanlarından biridir. Programın çalışırken karşılaşabileceği yükün test edildiği stres testi, programın güvenliğinin test edildiği güvenlik testleri ve çeşitli performans testleri gibi testlerden olan fonksiyonel olmayan test [22] grubunda da çalışmalar bulunmaktadır. Model tabanlı test için kullanılan sonlu durum makinalarındaki geçişlerin temsili ile modellenen durum tabanlı test verisi üretimi [23] de literatürde yapılmış çalışmalardandır.

Yazılımın test işlemlerinin başarımlarını ölçmede çeşitli metrikler kullanılmaktadır. Bu metriklerden bazıları [24];

- Fonksiyon kapsamı (Function coverage): Programdaki her bir fonksiyonun çağırılıp çağırılmadığı ile ilgilenir.
- İfade kapsamı (Statement coverage): Her bir satırın çalıştırılıp çalıştırılmadığı konusunda bilgi verir.
- Dal kapsamı (Branch coverage): Programdaki bütün olası dallanmaların çalıştırılıp çalıştırılmadığı hakkında bilgi verir.
- Yol kapsamı (Path coverage): Programa ait çalışması istenilen yolun ne kadarının çalıştırıldığını gösterir.

Meta-sezgisel algoritmaların kullanmış oldukları amaç fonksiyonları çeşitlilik göstermektedir. Çeşitli çalışmalarda kullanılan amaç fonksiyonları; kod/ifade kapsamı (code/statement coverage) [25], yol tabanlı kriter [26–33], kenar (edge) kapsamı [34], veri akış kapsamı (data flow coverage) [35], dal kapsamı (branch coverage) [36, 37], dal uzaklığı (branch distance) [38, 39], yakınsama seviyesi(approximation level) + yol uzaklığı (branch distance) [40] şeklinde verilebilir. Bu tez çalışmasında yakınsama seviyesi(approximation level) + dal uzaklığı (branch distance), benzerlik tabanlı ve yol tabanlı amaç fonksiyonları kullanılmıştır. Approximation level + branch distance metriklerini içeren amaç fonksiyonunda;

- Yakınsama seviyesi (Approximation level): Erişilmek istenen yola ne kadar yakın olunduğu hakkında bilgi verir.
- Dal uzaklığı (Branch distance): İlgili yola girmek için ne kadar uzakta olunduğu hakkında bilgi verir.

Benzerlik tabanlı amaç fonksiyonu çalıştırılması gereken ve çalıştırılan yollar arasındaki simetrik farka dayalı bir metriktir. Yol tabanlı ise çalıştırılması gereken ve çalıştırılan yolların birebir uyuşmasına dayanır. Bu metrikler hakkında ayrıntılı bilgi Bölüm 3'te verilmiştir.

Literatürdeki mevcut çalışmalar incelendiğinde; test verilerinin üretiminde çeşitli optimizasyon algoritmalarının kullanılabildiği, ayrıca yapılan pek çok çalışmada istenen sonuçları elde etme konusunda büyük eksiklikler bulunduğu anlaşılmıştır. Bazı algoritmalar istenen coverage miktarını sağlayamamışken yüksek coverage miktarına erişebilen algoritmalar ise hız konusunda yetersiz kalmıştır. Bu tez çalışmasında test verisi üretiminde literatürde henüz kullanılmamış güncel zeki optimizasyon algoritmaları ile literatürde kullanılan optimizasyon algoritmalarının başarımları daha önce bahsedilen metriklere dayalı olarak incelenecek ve bu algoritmaları performansları ve kullanılan amaç fonksiyonlarının başarımları karşılaştırılacaktır.

1. BÖLÜM

YAZILIM MÜHENDİSLİĞİ VE TEST

1.1. Yazılım Mühendisliği

Yazılım mühendisliği NATO Bilim Komitesinin 1968 ve 1969 yıllarında yapmış oldukları iki konferansta tartışılmış ve o zamandan beri sürekli popülerliği artan bir mühendislik alanı olmuştur [3]. Çoğu kişi bu konferansların tarihlerinin yazılım mühendisliğinin başlangıcı olduğuna inanmaktadır. Bu tarihten önce gerçeklenen yazılımlarda amaç bilgisayara hedeflenen işlemin yapılması idi. Yazılımların yapısı oldukça basit ve sade durumda nedeniyle yazılım mühendisliğine de ihtiyaç duyulmamaktaydı. Daha sonra bilgisayar donanımlarında gerçekleşen iyileştirmeler ve maliyetin düşmesi ile yaşanan bilgisayardaki kullanım sıklığı artışı geliştirilen yazılımların karmaşıklıklarının artmasına neden oldu. Günümüzde milyon satırlarla ifade edilen yazılımlar hayatımızın her alanında kullanılmaktadır.

Yazılım mühendisliği yazılımın gerçekleştirilmesindeki aşamalar olan tasarım, geliştirme, test ve bakım aşamalarını içine alan bir mühendislik dalıdır [3]. Yazılım mühendisliği alanı sürekli gelişmektedir ve geliştirme ile yönetim süreçlerini iyileştirmeye yönelik çeşitli teknikler önerilmektedir. Yazılım mühendisliği sayesinde karmaşıklığı artan yazılımların güvenirlik ve doğruluğu artırılmaktadır.

1.1.1. Yazılım Mühendisliği Nedir?

1969 yılındaki konferansta Fritz Bauer yazılım mühendisliğini "gerçek makineler üzerinde etkin ve güvenilir çalışan ekonomik yazılımlar geliştirilmesi amacıyla mühendislik ilkelerinin kullanımasıdır" diyerek tanımlamıştır [3]. Günümüzde yazılım

mühendisliği yazılım geliştirmenin her adımıyla ilgilenen bir mühendislik disiplinidir [41]. Hedeflenen zamanda hedeflenen bütçe ile hedeflenen işlevi yerine getirebilen yazılımlar başarılı yazılım projesi olarak adlandırılmaktadır. Yazılım mühendisliği de daha kaliteli, daha güvenilir ve başarılı yazılımların sayısını artırmayı amaçlamıştır.

1.1.2. Yazılım Geliştirme Fazları

Yazılımın geliştirilmesi sırasında gerçekleştirilen tanımlama (definition), ayrıntılandırma (elaboration), gerçekleştirme (construction), değerlendirme (evaluation), geçiş (transition) ve yönetim (management) aşamalarından oluşan fazlardan oluşmaktadır [41].

1.1.2.1. Tanımlama

Yazılım geliştirme sürecinin ilk aşaması tanımlamadır. Bu aşamada yazılımın ne yapacağı ve hangi probleme çözüm getireceği belirtilir. Nerede çalışacağı ve var olan kısıtlamalar belirlenmektedir. Bütün gereksinimler bu aşamada belirlenir. Bu aşamada belirlenen gereksinimler daha sonraki aşamalarda gerçekleştirilecek olan yazılımı şekillendirir. Bu yüzden geçerlilik, tutarlılık ve bütünlük gibi faktörler göze alınarak gereksinimler incelenir.

1.1.2.2. Ayrıntılandırma

Bu aşamada problemin yazılım ile nasıl çözülebileceği ve ortaya konan gereksinimlerin nasıl gerçekleştirilebileceği belirlenir. Yazılımın gerçekleştirilmesi için genel ve ayrıntılı modeller ortaya konur. Bu modeller gerçekleştirilecek olan yazılımın hangi alt sistemlerden oluşacağı ve sistemlerin nasıl bilgi alışveriş yapacağı hakkında bilgiler içerir. Bu aşama tasarım aşaması olarak da adlandırılır.

1.1.2.3. Gerçekleştirme

Bu aşamada yazılım kodlanır. Aynı zamanda kodlanan yazılımın her bir kod parçasının birim testleri de gerçekleştirilir. Daha sonra parçalar birleştirilerek entegrasyon testleri uygulanır.

1.1.2.4. Değerlendirme

Bir önceki aşamada gerçekleştirilen yazılımın belirlenen soruna çözüm getirip getirmediği değerlendirilir. Bu doğrultuda gerekli testler yapılarak raporlanır.

1.1.2.5. Geçiş

Değerlendirilmesi tamamlanan yazılımın sahaya sürülme aşamasıdır. Bu aşamadan sonra gelen geri dönüşlere göre yazılımda düzeltici, uyarlayıcı, iyileştirici veya önleyici bakımlar gerçekleştirilir.

Düzeltici Bakım: Daha önceden belirlenemeyen kusurların giderilmesi için gerçekleştirilen bakımdır.

Uyarlayıcı Bakım: Zamanla değişen donanım özelliklerine veya yasal düzenlemeler gibi çevresel etmenlere göre uyarlanması ile gerçekleştirilen bakımdır.

İyileştirici Bakım: İlk başta olmayan işlevlerin kazandırılması ile gerçekleştirilen bakımdır.

Önleyici Bakım: Yaşanan performans düşüşlerinin yeniden iyileştirilmesi ile gerçekleştirilen bakımdır.

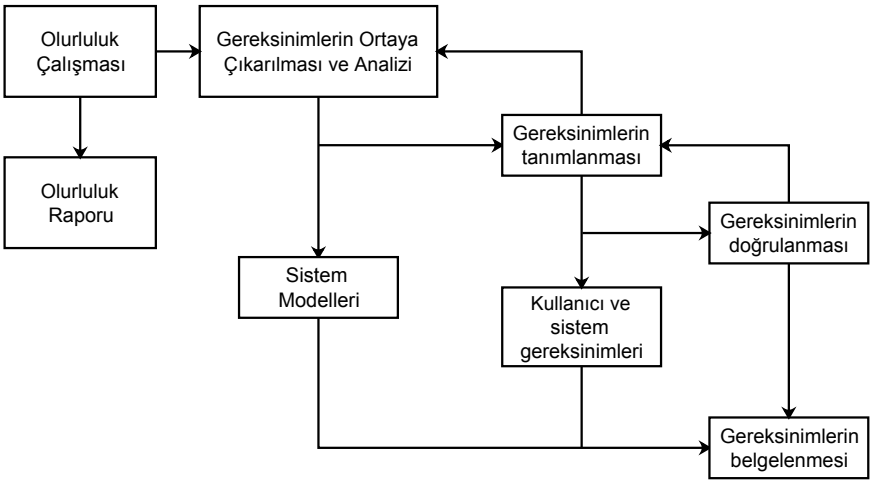
1.1.2.6. Yönetim

Tanımlama aşamasından geçiş aşamasına kadar bütün eylemlerin yönetim işlemi bu aşamada gerçekleştirilir.

1.1.3. Yazılım Gereksinim Belirtimleri

Yazılım gereksinimi, geliştirilecek olan yazılımdan ne beklendiğini ifade etmektedir. Amaca uygun yazılım geliştirilebilmesi için gereksinimlerin çok iyi belirlenmiş olması gerekmektedir. Bu nedenle gereksinimler Şekil 1.1'de görülen adımlar ile çıkarılmaktadır [3]. Aynı zamanda bu süreç gereksinim mühendisliği olarak da bilinmektedir [41].

Bu süreç;



Şekil 1.1. Gereksinim mühendisliği süreci. [3]

- Olurluluk çalışması ve raporlanması
- Gereksinimlerin ortaya çıkarılması ve analizi
- Gereksinimlerin tanımlanması
- Gereksinimlerin doğrulanması

aşamalarından oluşmaktadır. İlk olarak problemin anlaşılır ve kaynak harcanıp harcanmamasının gerekli olup olmadığı ile ilgili olurluluk çalışması yapılır. Daha sonra yazılıma neden ihtiyaç duyulduğu belirlenir. Kullanıcı, sistem ve yazılım açısından bakarak gereksinimler ortaya koyulur. Yazılım gereksinimlerinin ortaya konması başarılı yazılım projeleri için oldukça önemlidir. Çünkü bir yazılım projesinin doğru anlaşılması oldukça önemlidir. Bu gereksinimler;

- Doğruluk
- Açıklık
- Anlaşılırlık
- Yapılabilirlik
- Tutarlılık
- İzlenebilirlik

- Uyumluluk
- Test edilebilirlik

gibi kriterler ile gözden geçirilir. İhtiyacın doğru tespit edilmesi ve test edilmesi oldukça önemlidir çünkü bu aşamada yakalanan hataların düzeltilmesi çok daha kolay olacaktır.

1.1.4. Yazılım Süreç Modellerinde Yazılım Testleri

Yazılımlar insanlar tarafından gerçekleştirilen kodlardır. Bu nedenle yazılımlarda hata meydana gelmesi oldukça olasıdır. Meydana gelen bu hataların zamanında ortaya çıkarılması ve giderilmesi oldukça önemlidir. Yazılım süreç modellerinin amaçlarından biri de budur. Dünyadaki örnekler incelendiğinde yazılım projelerinin maliyetlerinin üçte birinden fazlası yazılım testlerine harcanmaktadır [3]. Yazılım testleri bu hataları bulmaya ve düzeltmeye odaklanır. Testler sırasında ne kadar çok hata bulunur ve düzeltilirse o kadar kaliteli bir yazılım ortaya konmuş olur.

1.2. Yazılım Testi

Yazılım mühendisliği dünyadaki en büyük sektörlerden biridir. Yazılım endüstrisi tarafından üretilen yazılımlar cep telefonlarından bilgisayarlara, sağlık sektöründe, askeri ve daha bir çok alanda karşımıza çıkmaktadır. Aynı zamanda bu alanlarda oldukça fazla ihtiyaç bulunmaktadır. Bu ihtiyaç doğrultusunda sürekli yeni projeler ortaya çıkmaktadır. Bu büyüme ile birlikte yazılımların karmaşıklıkları artmıştır. Artan bu karmaşıklık ise yazılımların kalitesini doğrudan etkilemiştir. Kaliteli yazılımlar elde etmek için yazılımın her aşamada test edilmesi gerekmektedir. Bu yüzden yazılım yaşam döngüsü içerisinde yazılımların kalitesini, güvenirliğini ve doğruluğunu etkileyen en önemli süreçlerden biri yazılım testleridir [3].

1.2.1. Yazılım Testi Nedir?

Yazılım testleri yazılım projelerinde üretilen ürünlerin daha az hata ile geliştirilmesini sağlayan aşamalardan biridir. Literatürde birden fazla yazılım testi ile ilgili tanım bulunmaktadır [3]. Bu tanımlardan bazıları;

- Yazılım testi, bir programın davranışını dinamik yöntemlerle, sonsuz bir küme içerisindeki belirli test durumlarını seçerek beklenen davranışa uyup uymadığını belirleme işlemidir [42].
- Bir sistemin veya bileşenin belirli koşullar altında çalıştırılması, sonuçların gözlenmesi veya kaydedilmesi ve belirli özelliklerin değerlendirilmesi sürecidir [43].
- Test, hata bulma amaçlı planlı bir şekilde gerçekleştirilen eylemler dizi ve doğrulama yöntemidir [44].
- Bir yazılım ögesinin mevcut ve olması gereken koşullar arasındaki farkın bulunarak analiz edilmesi ve yazılım ögesinin özelliklerinin değerlendirilmesi sürecidir [24].
- Test bir yazılım ürününün zayıf yönlerini veya makul hatalarını keşfetmek için gerçekleştirilen bir süreçtir [45].
- Input specificationları verildiğinde output specificationlarını üretebilmesinin kontrol edilmesidir.

1.2.2. Yazılım Testinin Önemi

Geçtiğimiz 30 yılda yazılım testlerinin önemi oldukça artmıştır. Artan programlama dilleri, farklılık gösteren işletim sistemleri ve donanımlar yazılım testinin uygulanmasını zorlaştırmıştır. Aynı zamanda bilgisayarlar her geçen gün daha fazla hayatımızın bir parçası olmakta ve farklı alanlarda kullanılmaktadır. Her geçen gün daha da güçlenen bir o kadar da fiziksel olarak küçülen bilgisayarlar televizyon, telefon, oyun konsolları gibi pek çok alanda aktif olarak görülmektedir.

Yazılımlarda meydana gelen bu hatalar para, zaman, itibar gibi kayıplara hatta yaralanma ve ölümlere neden olabilmektedir. Aşağıda bu yazılım hatalarından bazıları verilmiştir [3].

Denver Havaalanı Bagaj Sistemi: Zamanında dünyanın en büyük 2. havaalanı olan Denver havaalanında beklemeleri azaltmak ve daha az çalışan maliyetiyle daha hızlı ve doğru bagaj hizmeti sunmak için bir yazılım geliştirilmesi istemiştir. 186 milyon dolarlık bir proje olan bu bagaj sistemi yazılım hataları nedeniyle planlanan zamandan bir buçuk

yıl sonra devreye alınabilmiştir. Bu hataların neden olduğu zararın ekonomik boyutunun 340 milyon dolar olduğu hesaplanmaktadır.

Arienne 5 Füzesi: NASA uydu taşımak amacıyla 1996 yılında 7 milyar Euro'luk Arienne 5 isimli bir roket geliştirmiştir. 4 Temmuz 1996 yılında fırlatıldıktan kısa süre sonra havada patlayan bu roketin 16/64 bit çeviriminden kaynaklanan bir hata nedeniyle patladığı belirlenmiştir.

Patriot Füzeleri: 1991 yılındaki Körfez Savaşı sırasında Amerika'nın Suudi Arabistan'a yerleştirdiği patriot füzelerinden biri Irak'tan gelen Scud füzesini ıskalayarak 28 Amerikan askerinin ölümüne neden olmuştur. Yapılan incelemede füzede zaman hesaplamasında kullanılan 24 bitlik bir değişkende oluşan hata nedeniyle gerçekleştiği belirlenmiştir. Daha sonra yapılan araştırmaya göre bu hatanın daha öncesindeki yazılım güncellemesinde düzeltildiği fakat füzelerin güncellenmesinin ihmal edilmesi nedeniyle bu ıskalamayı yaptığı ortaya çıkmıştır.

Meydana gelen bu hataların başlıca nedenleri ise;

- İnsanların hata yapıp yanılabilen varlıklar olması
- Projenin tamamlanma süresinde bulunan zaman baskısı
- Altyapının karmaşık tasarlanmış olması
- Teknolojilerin sürekli değişim halinde olması
- Sistemlerin etkileşim halinde bulunup uyumsuz davranış göstermesi
- Radyasyon, manyetizma, elektrik alan gibi çevresel koşullar

olarak sıralanabilir. Nedenlerin bu kadar geniş çaplı olması ile birlikte hayatımızın her alanında yazılımların var olması nedeniyle hataların meydana gelme olasılığı da oldukça artmaktadır. Bu hatalar dikkatli bir şekilde test edilir ise yazılım uygulamaya geçilmeden önce düzeltilebilir. Bu sayede arıza oluşma riski azalır ve yazılımın kalitesi artar. Yazılım testi sayesinde yazılım hem fonksiyonel hem de fonksiyonel olmayan (güvenilirlik, kullanılabilirlik, verimlilik, sürdürülebilirlik, taşınabilirlik) gereksinimler açısından kalitesi de belirlenebilir. Test aşamasında bulunan hataların düzelmesi ile de

kalitesi artırılır. Yazılımın %100 test edilme şansı da bulunmamaktadır. Bu yüzden yazılımın test edilme süresine riskler, zaman ve bütçeye göre karar verilmelidir.

Yazılım testi yalnız yazılmış olan testlerin koşturulması olarak algılanmamalıdır. Test koşumu gerçekleştirilmeden önce ve sonra uygulanması gereken süreçler bulunmaktadır. Bu süreçler arasında;

- Planlama ve kontrol
- Test koşullarının seçilmesi
- Test senaryolarının tasarlanması ve koşturma
- Sonuçların kontrol edilmesi
- Çıkış değerlerinin değerlendirilmesi
- Test süreci ve test edilen sistem ile ilgili raporlama
- Test fazı sonlandırıldıktan sonra kapanış işlemleri ve tamamlama

bulunmaktadır [46]. Aynı zamanda doküman inceleme ve statik analizler de bu süreçlere dahil olan aşamalardan sayılabilir. Bu aşamalar gerçekleştirilirken hataları bulma, kalite seviyesini yükseltme, gelecekte meydana gelebilecek hataları önleme gibi hedefler bulunmaktadır.

1.3. Yazılım Test Düzeyleri

Yazılım testleri;

- Birim testler
- Yazılım entegrasyon testleri
- Sistem testleri
- Kabul testleri

olmak üzere dört farklı düzeyde gerçekleştirilmektedir [3].

1.3.1. Birim Testler

Birim testler kod parçacıklarının kendisinden beklenen işlevi yerine getirip getirmediğinin test edildiği testlerdir [24]. Başarılı ve kaliteli yazılım ürünleri ortaya çıkartmak için kullanılabilecek olan önemli bir test aşamasıdır. Yazılım projesinde başarılı olarak gerçekleştirilen birim testler sayesinde diğer düzeylerde gerçekleştirilecek olan testlerin de başarıya ulaşma şansı artmaktadır.

Birim testlerin gerçekleştirilmesi genelde bir test yazılımı aracılığı ile gerçekleştirilmektedir. Test yazılımının kullanılmadığı durumlarda ise kod geliştirme ile aynı anda birim testler geliştirilmektedir. Birim testler yazılım geliştirmenin bir parçasıdır ve buna göre birim testler de kodlanmalıdır.

Başarı ile gerçekleştirilen birim testler yazılımın kaliteli olduğunu göstermekle birlikte tamamen sorunsuz olduğunu göstermemektedir. Hemen ardından entegrasyon ve sistem testleri de gerçekleştirilmelidir.

Birim Testlerde Kapsama Analizi: Kapsama analizi önemli bir konudur. Test işleminde kapsama bir yazılımın içerisindeki tüm yol ve karar noktalarının test edilip edilmediğini belirtir. İki farklı durum barındıran bir kod parçası için testlerden birinin çalıştırılması demek %50 kapsamaya erişildiğini gösterir. Diğer test durumu da çalıştırılıp başarı ile sonuçlanırsa kapsama %100 olacaktır.

Birim Testlerde Sınır Değer Analizi: Birim testler gerçekleştirilirken sınır değerler de kontrol edilmelidir. Belirli değer aralıkları içerisinde işlem yapan metotların alt ve üst sınırlarını aşacak şekilde testler gerçekleştirilmelidir.

1.3.2. Entegrasyon Testi

Gerçekleştirilen yazılım projelerinde birim testlerin başarılı bir şekilde sonuçlandırılmasından sonra entegrasyon testleri başlamaktadır. Birim testlerinde her bir parçanın kendisinden beklenen davranışı gösterip göstermediği test edilirken entegrasyon testlerinde bir araya gelen bu modüllerin birlikte davranışlarının istenen şekilde olup olmadığı kontrol edilmektedir. Çünkü bu modüllerin birleştirilmesi sırasında veri kaybı yaşanabilmektedir. Aynı zamanda dikkatsizlik nedeni ile modüller birbirlerini

etkileyebilir, alt fonksiyonların birleştirilmesi ile beklenen sonuçlar elde edilemeyebilir veya birimlerdeki göze alınan hata toleranslarının birleşmesi ile genel veri yapılarında sorunlar yaşanabilmektedir [4]. Bu hata ve sorunların bulunup giderilebilmesi için entegrasyon testleri uygulanmalıdır. Entegrasyon testleri bütün olarak sınama (big bang) ve arttırmalı sınama olmak üzere iki farklı şekilde gerçekleştirilebilir. Big bang testi bütün modüller birleştikten sonra bütün olarak gerçekleştirilmektedir. Bu yöntem karışıklığa yol açabilmektedir bu yüzden sakıncalı olarak görülür. Arttırmalı sınama yönteminde ise modüller sırası ile eklenerek test işlemi gerçekleştirilmektedir.

1.3.3. Sistem Testleri

Sistem testleri; geliştirilen yazılımın performans, güvenilirlik, işlevsellik gibi özelliklerini değerlendiren testlerdir. Birim ve entegrasyon testleri gerçekleştirildiğinde yazılımın yapılmış olan tasarıma uygun geliştirildiği doğrulanmaktadır. Sistem testleri ise müşterini isteklerini karşılayıp karşılamadığının doğrulanmasını sağlar. Bu nedenle sistem test durumları geliştirilirken sistemin gereksinimlerini temel alarak sistemin çalışacağı gerçek ortamda karşılaşabileceği senaryolar test edilir.

Sistem testlerinde ilk olarak işlevsel gereksinimleri ne kadar sağladığı test edilmelidir. Bir sonraki aşamada ise işlevsel olmayan testler yapılır. İşlevsel olmayan testlerden bazıları:

Stres Testleri: Sistemde gerçekleştirilen zorlama durumlarında sistemin davranışının gözlendiği testlerdir.

Performans Testleri: Sistemden istenen çıktıların makul zamanda üretilip üretilemediği değerlendirilmesi yapılan testlerdir.

Konfigürasyon ve Uyumluluk Testleri: Sistemin farklı platform ve donanımlarda nasıl davrandığının tespit edildiği testlerdir.

Güvenlik Testleri: Sistemdeki yetkilendirme işlemlerinin sınandığı testlerdir.

Kullanılabilirlik Testleri: Kullanıcının sistem ile etkileşiminin sonuçlarının gözlendiği testlerdir.

Kullanıcı Arayüzü Testleri: Kullanıcının yazılımın arayüzü ile nasıl bir etkileşim

içerisinde bulunacağını ve kullanıcının klavye, ekran, fare ile sisteme vereceği komutları sistemin nasıl işleyeceğini değerlendirmek için yapılan testlerdir.

İşlevsel olmayan testler de tamamlandıktan sonra gerçekleştirilen sistem kullanıcı kabul testlerine hazır hale gelmektedir. Sistem testleri sırasında ortaya çıkan hatalar raporlanır ve gerekli düzeltme işlemleri gerçekleştirilir.

1.3.4. Kabul Testleri

Müşterinin sunmuş olduğu gereksinimler kabul testleri ile doğrulanmaktadır. Her bir kullanıcının gereksinimini doğrulayacak test durumları oluşturulur ve bu testler uygulanır. Sistemin bu testlerden geçmesi ile müşteri tarafından proje kabul görmüş olur.

Kabul testleri el ile yapılabildiği gibi otomatik olarak da gerçekleştirilebilmektedir. Kabul testlerinin sistemin başarısını doğrudan göstermesinden dolayı ayrı bir önem arz etmektedirler. Bu testlerdeki en büyük sorun testlerin kim tarafından nasıl hazırlanacağıdır. Kabul testleri doğrudan müşteri tarafından yazılabileceği gibi test ekibi tarafından da yazılabilir. Eğer test ekibi testleri hazırlar ise müşteri de bu testlere onay vermelidir. Bazı projelerde ise bu testleri ortak bir ekip hazırlamaktadır.

1.4. Yazılım Test Teknikleri

Yazılım testleri kara kutu, saydam kutu ve gri kutu olmak üzere üç farklı teknikte gerçekleştirilmektedir.

1.4.1. Kara Kutu Testi

Kara kutu testleri yapılır iken yazılımın iç yapısı hakkında bilgiye ihtiyaç duyulmamaktadır. Kara kutu testinde sisteme uygulanan girişe karşılık istenen çıkışın elde edilip edilmediği sınanmaktadır. Bu test yöntemini yazılımcının kendisinden daha çok test mühendisleri kullanmaktadır. Kara kutu testinin kullanılma amaçları [3];

1. Doğru olmayan veya hiç olmayan işlevlerin tespiti
2. Arayüz hataları

3. Performans hataları
4. Veri tabanlarına ulaşma hataları veya veri yapılarındaki hatalar
5. İlklendirme veya sonlandırma hataları
6. Sınır değer hataları

olarak sıralanabilmektedir.

Bu test tekniği ile yazılımlarda bulunan hatalar hızlı bir şekilde bulunabilmektedir. Aynı zamanda test durumları yapılmış olan gereksinim analizlerine bağlı olarak hazırlanmaktadır. Bu yüzden gereksinimlerdeki tutarsızlık ve belirsizlikler belirlenebilmektedir. Yazılımın iç ayrıntısını testi gerçekleştirecek olan kişinin bilmesine de gerek yoktur. Kara kutu testi kodu yazacak insan ile testi yapacak insanın da farklı olabilmesini sağlar. Aynı zamanda bu yazılım tekniği yazılıma kullanıcı gözü ile baktığı için kodu yazan insanların fark edemeyeceği pek çok hatanın bulunmasına yardımcı olmaktadır. Testi gerçekleştirecek kişilerin sistem hakkındaki teknik bilgileri bilmesine de gerek yoktur. Bunlar kara kutu testinin avantajlarıdır. Kara kutu testlerinin bu avantajlarının yanında çeşitli dezavantajları da bulunmaktadır. Yazılıma bütün olarak yaklaştığı için pek çok hata tespit edilemeden kalabilmektedir. Testin gerçekleştirilmesi için kullanılan girdiler de kısıtlıdır. Bütün kombinasyonların denenmesi oldukça zordur bu yüzden pek çok durum test edilemeden kalmaktadır. Aynı kod parçacıkları çok defa test edilirken bazı parçalar ise hiç test edilemeden kalabilmektedir. Tam olarak belirtilememiş gereksinimlerin test durumlarının tasarlanması ve test edilmesi bu teknik ile oldukça zordur.

1.4.2. Saydam Kutu Testi

Saydam kutu testi kara kutu testinin aksine yazılımın iç yapısına göre tasarlanmaktadır. Bu yüzden genellikle bu testi yazılımı gerçekleştirenler yapmaktadır. Bu testte programın içindeki birimlerin doğru çalışıp çalışmadığı araştırılmaktadır. Bu yüzden yazılımın kaynak koduna ihtiyaç duyulmaktadır. Saydam kutu testleri veri, kontrol ve bilgi akışlarının, kodlama standartlarının, hata yakalama ve ayıklama yapısının analizlerini içermektedir [3]. Bu yaklaşım ile yapılan testler;

Birim Testler: Saydam kutu testinin en çok kullanıldığı testler birim testlerdir. Kodun belirli bir parçasının istenen görevi yapıp yapamadığı test edilmektedir.

Statik ve Dinamik Analizler: Statik analiz, yazılmış olan kodun incelenerek yapıldığı ve hataların tespit edilmeye çalıştığı yöntemdir. Dinamik analiz ise kod parçasının çalıştırılarak analiz edildiği yöntemdir.

Deyim Kapsama (Statement Coverage): Yazılmış olan kod parçasının her bir satırının çalıştırılarak test edildiği ve bu satırların herhangi bir probleme sebep olup olmadığının belirlendiği yöntemdir. Bu sayede kod içerisinde çalıştırılmayan satır olup olmadığı da doğrulanır.

Dal Kapsama (Branch Coverage): Kod içerisindeki karar noktalarının ve bu karar noktaları ile ayrıldığı dalların çalıştırılıp davranışının analiz edildiği yöntemdir. Dal kapsama ile program içerisinde yer alan tüm dalların kendilerinden beklenildiği şekilde çalıştığı doğrulanır.

Yol Kapsama (Path Coverage): Kod içerisindeki bütün yolların test edildiği yöntemdir.

Saydam kutu testleri ile yapılmış olan hatalar hızlı bir şekilde tespit edilebilmektedir. Bu tespit edilen hatalar düzeltilerek entegrasyon ve sistem testlerinin daha başarılı olması sağlanabilmektedir.

Saydam kutu testleri sayesinde kod içerisindeki mantıksal hatalar bulunabilmektedir. Yazılan kodun en iyilenmiş bir şekilde yazılmış olmasını da sağlamaktadır. Kod içerisindeki çalışmayan fazla satırların da bulunarak temizlenmesini sağlar. Kaynak kod içerisinde gerçekleştirilmesi nedeniyle hataların da erken aşamalarda bulunmasını sağlar. Aynı zamanda bu test tekniği yazılımcının da kendini geliştirmesini ve kod yazma yeteneğinin artmasını sağlar. Saydam kutu testleri ile sadece birimlerin iç işleyişi test edileceği için modüllerin birleşmesinden sonra çıkabilecek hataları tespit edememektedir.

1.4.3. Gri Kutu Testi

Gri kutu testlerinde saydam ve kara kutu testleri birlikte kullanılmaktadır. Bu testlerde gereksinimlere göre uygun olarak test durumları kodun iç yapısına uygun olarak

tasarlanmaktadır. Böylece hem gereksinimlerin gerçekleştirilmiş olduğu doğrulanır hem de kodun iç yapısı test edilmiş olur. Gri kutu testinin dezavantajı ise test ekibinin iç yapısını bilmesi nedeniyle koda karşı gerçekleşecek bir şartlanmanın bazı hataların gözden kaçmasına sebep olabilmesidir.

2. BÖLÜM

ARAMA TABANLI TEST VERİSİ ÜRETİMİ

2.1. Arama Tabanlı Test Verisi Üretimi

Arama tabanlı test verisi üretiminde (SBST) temel amaç kapsama metriğini maksimum yapacak test verisinin üretiminin sağlanmasıdır. Bu amaçla rastgele test verisi üretimi yöntemi [47] sıkça kullanılmakta ve düşük maliyetli kolay kısıtlara sahip kod parçaları için verilen değer aralığında test verileri üretmektedir ancak rastgele test verisi üretimi yöntemi zor kısıtlı problemlerde oldukça kötü performans göstermektedir.

Arama tabanlı test verisi üretimi ayrık bir problem olarak sınıflandırılmaktadır ve bu problemin çözümünde meta-sezgisel algoritmalar kullanılabilmektedir. Harman ve Jones [11] yazılım mühendisliği problemi olan test verisi üretiminde meta-sezgisel algoritmalar kullanımının kapsama metriğine bağlı olarak 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 yüzden test verisi üretiminde de kullanımları oldukça yaygındır. Arama tabanlı test verisi üretimi hakkında yapılan çalışmalar Harman ve Jones (2001) [11], McMinn (2004) [48], Afzal ve ark. (2009), Räihä (2010) [49], McMinn (2011) [13], Harman ve ark. (2012a) [50], Harman ve ark. (2012b) [51] tarafından derlenmiştir. Bütün bu derlemelerden anlaşıldığı üzere arama tabanlı yazılım mühendisliği alanı her geçen gün daha da popüler hale gelmektedir.

Doğadaki olaylardan esinlenen pek çok meta-sezgisel algoritma literatüre sunulmuştur. Parçacık sürü optimizasyonu (PSO) [52], diferansiyel gelişim (DE) [53], yapay arı koloni (ABC) [54] ve ateş böceği (FA) algoritmaları [55] en popüler meta-sezgisel algoritmalar

arasında gösterilmektedir.

PSO algoritması [52] Eberhart ve Kennedy tarafından 1995 yılında literatüre sunulmuş kuş ve balık sürülerini model alan sürü tabanlı meta-sezgisel algoritmadır. DE algoritması [53] Storn ve Kennedy nümerik optimizasyon problemleri için literatüre tanıtılmış, çaprazlama, mutasyon ve seleksiyon operatörlerinden oluşan meta-sezgisel bir algoritmadır. ABC algoritması [54] arıların yiyecek arana davranışını modelleyen bir algoritmadır. 2005 yılında Karaboğa tarafından literatüre sunulan ABC algoritması hakkında çeşitli alanlarda çalışmalar bulunmaktadır [56–58]. FA algoritması [55] ise ateş böceklerinin birbirleri arasındaki iletişimini temel alan ve 2008 yılında Yang tarafından literatüre sunulan meta-sezgisel bir algoritmadır.

Son yıllarda PSO, DE, ABC ve FA algoritmaları ile test verisi üretimi hakkında yapılmış çalışmalar bulunmaktadır. Windisch ve ark. [36] PSO algoritması ile branch coverage kriterini birlikte kullanarak test verisi üretmiş ve elde edilen sonuçlar Genetik Algoritma (GA) ile kıyaslamıştır. Elde edilen sonuçlara göre PSO algoritması GA'ya göre daha iyi sonuçlar üretmiştir. Tiwari ve ark. [25] PSO'nun bir türevini regresyon testi için kullanmıştır. Beş benchmark üzerinde yapmış olduğu çalışmaya göre önermiş olduğu PSO türevi algoritma diğer PSO türlerine göre daha iyi kapsama sonuçları üretmiştir. Zhu ve ark. [38] eylemsizlik ağırlığı (w) değerinin adaptif olarak değiştiği yeni bir PSO türü önermiştir. Elde edilen sonuçlara göre önerilen yeni model GA ve PSO algoritmalarından daha iyi sonuçlar üretmiştir. Dahiya ve ark. [29] C programları üzerinde bütün yolların test edilmesine dayalı (all-path testing) hibrit sözde dinamik PSO modeli önermiştir. Çalışma yapısal test problemleri üzerinde gerçekleştirilmiş ve önerilen tekniğin daha gürbüz olduğu belirtilmiştir. Singla ve ark. [35] test verisi üretimi için farklı boyutta ve karmaşıklıkta problemler için GA ve PSO algoritmaların birleşiminden oluşan bir model önermişlerdir. Önerilen bu yeni model hem GA hem de PSO algoritmalarından daha iyi sonuçlar vermiştir. Latiu ve ark. [40] approximation level $+$ branch distance metriklerini kullanarak GA, PSO ve ısıl işlem (SA) algoritmalarını çalıştırmıştır. Sonuçlar meta-sezgisel algoritmaların test verisi üretiminde etkin olarak kullanılabileceğini göstermiştir.

Landa Becerra ve ark. [37] dal kapsamı (branch coverage) metriği ile DE algoritmasının

çeşitlerini kullanarak test verisi üreticisi ortaya koymuşlardır. Temel DE algoritması Breeder DE ile kıyaslanmış ve DE algoritmasının gerçek dünya problemleri için daha uygun olduğu ifade edilmiştir. Jianfeng ve ark. [59] DE tabanlı kombinasyonel test veri üreticisi oluşturmuşlardır. Önerilen yaklaşım otomatik test veri üreticisi, SA, GA, karınca koloni (ACO), Cross-Entropy ve PSO algoritmaları ile kıyaslanmış ve önerilen yaklaşımın da iyi sonuçlar üretebildiği belirtilmiştir. Liang ve ark. [60] DE tabanlı tek seferde tek test stratejisine dayalı test durumu üreticisi önermiştir. Çalışmada farklı mutasyon operatörleri ile kontrol parametre değerleri analiz edilmiştir. Önerilen yaklaşımın iyi sonuçlar üretebildiği söylenmiştir.

Mala ve ark. ABC algoritmasını test verisi üretmek için kullanmıştır. ABC ve GA [26] ile ABC ve ACO [27] kıyaslamaları yapılmıştır. Bu çalışmalarda kapsama tabanlı amaç fonksiyonu kullanmışlardır. Önerilen yöntemin daha az hesaplama zamanına ihtiyaç duyduğu, daha ölçeklenebilir ve verimli sonuçlar ürettiği görülmüştür. Yaptıkları bir başka çalışmada ise [28] hesaplama zamanını düşürmek için paralel ABC algoritmasını kullanmışlardır. Kullandıkları ABC algoritmasını ABC, GA ve RS algoritmaları ile kıyaslamışlardır. Sonuçlar önerilen yaklaşımın daha kısa sürede sonuca ulaşabildiğini göstermiştir. Dahiya ve ark. [39] branch distance tabanlı amaç fonksiyonu kullanarak yapısal test verisi üretimi için bir ABC modeli önermişlerdir. Önerilen bu model geniş uzayda değerler alabilen 10 gerçek dünya problemi üzerinde çalıştırılmıştır. Elde edilen sonuçlara göre önerilen yöntem parametre arama uzayı büyüdüğü ve incelenen kod parçalarında çok fazla kısıt olduğu zaman iyi sonuçlar üretmektedir. Lam ve ark. [34] daha az sayıda koşma ile sonuç üretmeye dayalı paralel ABC modeli önermişlerdir. Elde edilen sonuçlara göre önerilen çalışma bölgesel optimumlara takılmamakta ve literatürdeki pek çok amaç fonksiyonundan daha iyi sonuçlar üretmektedir. Malhotra and Khari [30] GA algoritmasındaki mutasyon operatörünün kullanıldığı bir ABC türünü kullanmıştır. Önerilen yöntem 10 adet C++ programı üzerinde çalıştırılmış ve ABC algoritmasından daha iyi sonuçlar ürettiği gözlenmiştir. Malhotra ve ark. [31] yol kapsama metriği tabanlı amaç fonksiyonu kullanarak ABC, ACO ve GA algoritmalarını 9 adet C++ programı üzerinde çalıştırmışlardır. Sonuçlara göre ABC algoritması diğer algoritmalardan daha başarılı olmuştur. Suri ve Kaur [32] ABC algoritmasını regresyon testi için kullanmıştır. Yapılan bu çalışma programın etkilenen yüzdesini bulmakta ve

maksimum yol kapsama miktarına erişmeye çalışmaktadır. Yeni test durumları kapsama miktarı %100 olana kadar devam etmektedir. Çalışma 8 örnek üzerinde yapılmış ve %100 kapsama miktarına erişilmiştir.

Srivatsava ve ark. [33] optimize edilmiş test durumları ve test etmek için yollar üreten FA tabanlı bir model önermiştir. Önerilen bu model matris tabanlı bir amaç fonksiyonu kullanarak yollarda indirgeme yapmış ve test edilebilmesi için optimum sayıda test yolları üretmiştir.

2.2. Popülasyon Tabanlı Meta-sezgisel Algoritmalar

Gerçek dünya problemleri genellikle zor ve karmaşıktır. Bu yüzden de problemlerin çözümü ya elde edilememekte ya da çözümün elde edilebilmesi için oldukça fazla zaman gerekmektedir. Sezgisel algoritmalar bu zor problemlerin çözümü için geliştirilen yöntemlerdir ve bir çok tasarım probleminin çözümünde kullanılmaktadır. Bu algoritmaların genel özellikleri; en iyi veya en iyiye yakın çözümü elde etmeleri, non-deterministik olmaları, yerel optimumlardan kurtulmak için çeşitli stratejilere sahip olmaları ve probleme özgü olmamalarıdır [61–63]. Popülasyon tabanlı sezgisel algoritmalar daha genel ve başarılı yeni algoritma modelleri ortaya koymak veya mevcut algoritmaların başarısını artırmak amacıyla araştırmacıların ilgisini çeken bir konu olmuştur. Popülasyon tabanlı sezgisel algoritmalar başlangıçta rastgele oluşturdukları popülasyonu iteratif olarak iyileştirerek makul sürede en iyi veya en iyiye yakın çözüme ulaşmaya çalışan algoritmalardır [64]. Bu algoritmalarda başlangıç popülasyonu alt ve üst sınıra bağlı olarak rastgele üretilir. Başlangıç popülasyonundaki çözümlerin üretilmesinde Eşitlik 2.1 kullanılabilir.

$$x_{ij} = x_j^{min} + rand()(x_j^{max} - x_j^{min}) \tag{2.1}$$

i ve j indisleri sırasıyla popülasyondaki birey ve optimize edilecek parametreyi göstermektedir.

2.2.1. Parçacık Sürü Optimizasyonu

Parçacık sürü optimizasyon algoritması kuş ve balık sürülerinin iki boyutlu hareketinden esinlenilerek Kenndey ve Eberhart tarafından geliştirilmiş popülasyon tabanlı sezgisel algoritmadır [52]. Algoritma kuşların ve balıkların yiyecek kaynaklarına doğru yaptıkları yönelimi benzetmektedir. Algoritmada her bir aday çözüm parçacık ve popülasyon ise sürü ile ifade edilir. Öncelikle başlangıç popülasyonu rastgele oluşturulur ve her bir iterasyonda bu çözümler iyileştirilmeye çalışılır. İyileştirme işlemi o popülasyondaki en iyi çözüm ($gbest$) ve kendi en iyi çözümü ($pbest$) kullanılarak gerçekleştirilir. Bu sayede bireyler arasında bilgi aktarımı olmaktadır. PSO algoritmasının temel adımları Algoritma 2.1'da verilmiştir.

1: Başlangıç popülasyonunu oluştur
2: Uygunluk değerini hesapla
3: **repeat**
4: En iyi çözümü belirle ($gbest$)
5: Kendi en iyi değerini güncelle ($pbest$)
6: Hız vektörünü oluştur
7: Yeni aday çözüm üret
8: Güncelle
9: **until** Durdurma kriteri sağlanana kadar

Algoritma 2.1: PSO algoritmasının temel adımları.

Kendi en iyi değeri ($pbest$) ve popülasyondaki en iyi değerin ($gbest$) bulunmasının ardından hız vektörü ve yeni çözüm Eşitlik 2.2 ve 2.3 ile güncellenir.

$$\begin{aligned}\vec{v}(t+1) = \quad & \omega\vec{v}(t) + \phi_1 \mathrm{rand}(0,1)(\vec{pbest}(t) - \vec{x}(t)) \\ & + \phi_2 \mathrm{rand}(0,1)(\vec{gbest}(t) - \vec{x}(t))\end{aligned} \tag{2.2}$$

Burada ω hız değerinin etkisini belirleyen ağırlık, ϕ_1 ve ϕ_2 sosyal ve bilişsel öğrenme faktörleridir. ϕ_1 ve ϕ_2, her parçacığı $pbest$ ve $gbest$ pozisyonlarına doğru ilerlemesini sağlayan sabitlerdir. ϕ_1, parçacığın kendi bilgisine göre hareket etmesini, ϕ_2 ise sürüdeki diğer parçacıkların bilgisine göre hareket etmesini sağlar. Bu değerler düşük seçilir ise araştırma uzayının yavaş, yüksek seçilmesi ise hızlı taranmasını sağlar. Eğer yavaş taranırsa sonucun elde edilmesi de yavaş olacaktır. Hızlı taranması durumunda ise optimum bölgenin atlanmasına neden olabilir. Yeni hız değeri ile mevcut pozisyon Eşitlik

2.3 ile güncellenir.

$$\vec{x}(t+1) = \vec{x}(t) + \vec{v}(t+1) \tag{2.3}$$

2.2.2. Yapay Arı Koloni Algoritması

Yapay arı koloni algoritması arıların doğadaki yiyecek arama davranışlarını modelleyen ve 2005 yılında Karaboğa tarafından literatüre sunulmuş popülasyon tabanlı sezgisel algoritmalardan biridir [54]. Yiyecek arama işinde iş bölümü ile arılar görevli, gözcü ve kaşif arılar olmak üzere üç farklı gruba ayrılırlar. Görevli arılar hafızalarındaki kaynaklar etrafında araştırma yapmaktadır. Bu da yerel arama işlemine denk gelmektedir. Gözcü arılar görevli arılardan alınan bilgilerle potansiyel olarak daha iyi yiyecek kaynaklarını seçerler. Kaşif arılar ise keşfedilmemiş rastgele kaynakları ortaya çıkarmak ile sorumludur. Bu da global arama yeteneğini modellemektedir. ABC algoritmasının temel adımları Algoritma 2.2'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 2.2: ABC algoritmasının temel adımları.

Bu algoritmada kontrol parametreleri belirlendikten sonra yiyecek kaynakları pozisyonlarına karşılık gelen başlangıç popülasyonu alt ve üst sınır aralığında rastgele olarak oluşturulmaktadır. Daha sonra bu popülasyonun bireyleri amaç fonksiyonu yerine koyularak uygunluk değerleri hesaplanmaktadır. Görevli arı aşaması, çözümlerin civarının daha iyi araştırılmasını sağlar. Her bir çözüm vektörünün komşulukları Eşitlik 2.4'de verilen denklem ile aranır.

$$x'_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) \tag{2.4}$$

Eşitlik 2.4'de x' yeni aday çözüm vektör, ϕ $[-1, 1]$ aralığında üretilen rastgele bir değer, x_k rastgele seçilmiş bir komşu vektördür. Üretilen çözüm sonrası açgözlü seçim metodu

kullanılarak yeni çözüm ve mevcut çözüm arasında daha iyi olanı seçmeye dayalı bir tercih yapılır. Yeni çözüm mevcut çözümden daha iyi ise yeni çözüm seçilir, aksi takdirde mevcut çözüm korunur ve daha iyi bir çözüm elde edilemediği için daha iyi çözümün elde edilememesinin sayıldığı sayaç bir artırılır. Bu sayaç kâşif arı biriminde yiyecek kaynağının bittiğine karar vermede kullanılır. Gözcü arı aşamasında da yeni çözüm üretmek için Eşitlik 2.4 kullanılmaktadır. İyileştirilme yapılacak olan çözüme karar vermek için iyi çözümlerin seçilme olasılığının daha yüksek olduğu olasılıksal seçim metodu kullanılmaktadır. Bu da iyi bireylerin seçilme şansını artırarak iyi çözümlerin etrafında yerel arama yapılmasını sağlamaktadır. Ancak kalite değeri düşük çözümlerin civarının araştırılmasına da şans tanımaktadır. Olasılık seçimi için Eşitlik 2.5 kullanılmaktadır.

$$p_i = \frac{uygunluk_i}{\sum_{i=1}^{NP} uygunluk_i} \tag{2.5}$$

Burada p_i, i. çözümün seçilme olasılığı ve $uygunluk_i$ çözümün kalitesine karşılık gelmektedir. Görevli arılarda olduğu gibi burada da açgözlü seçim yöntemi ve iyi çözüm vermeyen yerel arama sayısını tutan sayaçlar kullanılmaktadır.

Doğada arıların nektar toplaması sonucu zamanla yiyecek kaynakları tükenmektedir. Algoritmada ise bu çözümün artık iyileştirilemiyor olmasına karşılık gelmektedir. Kaynağın yeterince aranıp aranmadığı "limit" adı verilen kontrol parametresi ile belirlenmektedir. x_i konumundaki çözümün "limit" geliştirilememe sayacının değerini aşmış ise x_i terk edilir ve o kaynağın arısı artık kaşif arı haline gelerek rastgele araştırma yapmaktadır.

2.2.3. Diferansiyel Gelişim Algoritması

Diferansiyel gelişim algoritması popülasyon tabanlı sezgisel algoritmalardan biridir ve 1995 yılında geliştirilmiştir [53]. DE algoritması gelişime dayalı algoritmalardaki gibi mutasyon, çaprazlama ve seleksiyon operatörlerini kullanmaktadır. DE algoritmasının temel adımları Algoritma 2.3'de gösterilmiştir.

Kontrol parametreleri belirlenip başlangıç popülasyonu oluşturulduktan sonra bu

1: Başlangıç popülasyonunu oluştur
2: Uygunluk değerlerini hesapla
3: **repeat**
4: Mutasyon
5: Çaprazlama
6: Seleksiyon
7: **until** Durdurma kriteri sağlanana kadar

Algoritma 2.3: DE algoritmasının temel adımları.

popülasyona ait bireylerin uygunluk değerleri hesaplanmaktadır. Mutasyon aşamasında mevcut vektörün parametre değerleri çözüm vektörleri arasındaki farktan yararlanarak Eşitlik 2.6 ile değiştirilmektedir. Burada değişikliğe uğrayacak çözüm vektörü ve popülasyon dışında üç farklı çözüm vektörü alınmaktadır. Bunlardan ikisinin farkı alınıp ölçeklendikten sonra diğer vektöre eklenir.

$$\hat{x}_i = x_{r_1} + F(x_{r_3} - x_{r_2}) \tag{2.6}$$

Burada r_1, r_2, r_3, $[1, NP]$ aralığında her bir iterasyonda rastgele seçilen birbirinden farklı tamsayılardır. Ölçekleme faktörü F ise genellikle $[0, 1]$ aralığında seçilir. $\hat{x}_i$ ise oluşturulan fark vektörüdür.

Çaprazlama aşamasında mutasyonda oluşan fark vektörü $\hat{x}_i$ ile x_i vektörü kullanılarak Eşitlik 2.7 ile yeni aday çözüm elde edilir.

$$y_i^j = \begin{cases} \hat{x}_i^j & R_j \leq CR \\ x_i^j & R_j > CR \end{cases} \tag{2.7}$$

Burada CR çaprazlama oranıdır. Her parametre için $(0, 1)$ aralığında rastgele değerler üretilir ve CR'den düşükse $\hat{x}_i$ aday çözümüne ait parametre değeri yoksa x_i çözümünün parametre değeri alınır.

Seleksiyon aşamasında oluşturulan aday çözüm ile değişikliğe uğrayacak çözüm arasında Eşitlik 2.8 ile açgözlü seçim metodu kullanılarak seçim işlemi yapılır. Bu seçim işlemi uygunluk fonksiyonu değerleri karşılaştırılarak karar verilir.

$$y_i = \begin{cases} \hat{x}_i & f(\hat{x}_i) \leq f(x_i) \\ x_i & diger \end{cases} \tag{2.8}$$

y_i popülasyondaki yeni birey ve $f(\hat{x}_i)$ ile $f(x_i)$ sırasıyla aday çözüm vektörü ile değişikliğe uğrayan vektörün uygunluk değerleridir.

2.2.4. Ateş Böceği Algoritması

Ateş böceği algoritması, ateş böceklerinin sosyal davranışlarından esinlenilerek Yang tarafından geliştirilmiş popülasyon tabanlı sezgisel algoritmadır [55]. Ateş böceği sürüsü popülasyon olarak düşünülmektedir ve popülasyondaki bireylerin ışıklarını yakıp söndürmesi ile diğer bireyleri kendine çekmesi prensibine dayanmaktadır. Popülasyonun daha parlak yani çekici yerlere gitmesi ışık şiddetine bağlıdır. Popülasyondaki bireyler parlaklıklarına bağlı olarak birbirlerini çekmektedir. Ateş böceği ne kadar parlak olursa çözüm de o kadar kalitelidir ve diğer ateş böcekleri için de o kadar çekicidir. Bununla beraber mesafe parlaklığı azaltacağı için çekim kuvvetini de azaltmaktadır. FA algoritmasının temel adımları Algoritma 2.4'de gösterilmiştir.

1: Başlangıç popülasyonunun oluşturulması
2: Uygunluk değerlerinin hesaplanması
3: **repeat**
4: Bütün ateş böceklerinin parlaklıklarının hesaplanması
5: Parlaklığa göre ateş böcekleri ile etkileşimin gerçekleşmesi
6: Arama uzayında hareketin gerçekleşmesi
7: Yeni ateş böceklerinin oluşturulması ve parlaklık yoğunluklarının güncellenmesi
8: **until** Durdurma kriteri sağlanana kadar

Algoritma 2.4: FA algoritmasının temel adımları.

Ateş böceği algoritmasında temel nokta ışık yoğunluğunda yaşanan değişim ve bireylerinin çekiciliğine dayanmaktadır. Çekicilik değeri (β) görecelidir ve ateş böceği i ile ateş böceği j arasındaki mesafe olan r_{ij} değerine bağlı olarak değişecektir. Dolayısıyla ışık kaynağından uzaklaştıkça şiddeti azalacaktır. Işık şiddeti, r mesafe, I_s ise ışık şiddeti olmak üzere mesafenin karesine bağlı olarak Eşitlik 2.9'de gösterilmiştir.

$$I(r) = \frac{I_s}{r^2} \tag{2.9}$$

İki ateş böceği arasındaki mesafe Eşitlik 2.10'daki denklem ile, yeni çözüm üretimi ise Eşitlik 2.11 ile gerçekleştirilebilir.

$$r_{ij} = \sqrt{\sum_{k=1}^{D}(x_{ik} - x_{jk})^2} \tag{2.10}$$

$$x_i^{t+1} = x_i^t + \beta_0 e^{-\gamma r_{ij}^2}(x_j^t - x_i^t) + \alpha_t \epsilon_i^t \tag{2.11}$$

Burada $j \in [1, NP]$ aralığında $j \neq i$ olmak kaydıyla komşu bir çözüm vektörü, α adım büyüklüğü, ϵ_i Gauss veya uniform dağılımlı bir vektör, γ ateş böcekleri arasındaki çekicilik değişimlerini belirleyen bir sabit, r ateş böcekleri arasındaki mesafe ve β_0'da onların çekiciliğini göstermektedir.

3. BÖLÜM

META-SEZGİSEL ALGORİTMALARIN TEST VERİSİ ÜRETİMİNDE KULLANILMASI

Bu tez çalışmasında; temel ve döngü içeren problemler olmak üzere iki farklı grupta 10 farklı problem üzerinde PSO, DE, ABC ve FA algoritmaları ile RS algoritması uygulanmıştır. PSO, DE, ABC ve FA algoritmaları diğer sezgisel algoritmalara göre uygulaması kolay ve verimli sonuçlar üretebilen algoritmalardır. Özellikle ABC algoritmasının az kontrol parametresine sahip olması nedeniyle kontrol parametrelerinin belirlenmesi kolaydır. Çalışmanın ilk aşamasında temel problemler incelenmiştir. Temel problemlerde; literatürde incelenen diğer çalışmaların büyük bir çoğunluğunda kontrol parametrelerinin analizi bulunmamaktadır. Algoritmaların yerel ve global arama yetenekleri kontrol parametrelerine oldukça bağımlı olduğundan çalışmanın ilk aşamasında parametre analizi yapılmıştır.

Meta-sezgisel algoritmalar amaç fonksiyonları doğrultusunda arama yaparlar. Bu yüzden amaç fonksiyonunun tasarımı da oldukça önemlidir [11]. İyi bir amaç fonksiyonu algoritmanın optimum değeri hızlı ve doğru bir şekilde bulmasına yardımcı olur. Çeşitli çalışmalarda kullanılan amaç fonksiyonları; code/statement coverage [25], yol tabanlı kriter [26–33], kenar (edge) kapsamı [34], veri akış kapsamı (data flow coverage) [35], dal kapsamı (branch coverage) [36, 37], branch distance [38, 39], approximation level + branch distance [40] şeklinde verilebilir. Amaç fonksiyonları algoritmaların performansını etkilediği için farklı amaç fonksiyonlarının analizi önemli bir husustur. Bu nedenle bu çalışmada en sık kullanılan amaç fonksiyonlarının analizi belirlenen parametrelerle yapılmıştır.

Üçüncü olarak belirlenen parametre değerleri kullanılarak meta-sezgisel algoritmaların test verisi üretiminde performans kıyaslaması yapılmıştır.

Son olarak ise meta-sezgisel algoritmaların performansları ile rastgele arama (RS) algoritmasının performansı karşılaştırılmıştır.

Temel problemlerden sonra döngü içeren problemler ele alınmıştır. Bu problemlerin kontrol parametreleri analiz edildikten sonra elde edilen parametre değerleri ile meta-sezgisel algoritmaların farklı amaç fonksiyonlarındaki performansları incelenmiştir. Son olarak ise amaç fonksiyonlarının kendi aralarında kıyaslanması yapılmıştır.

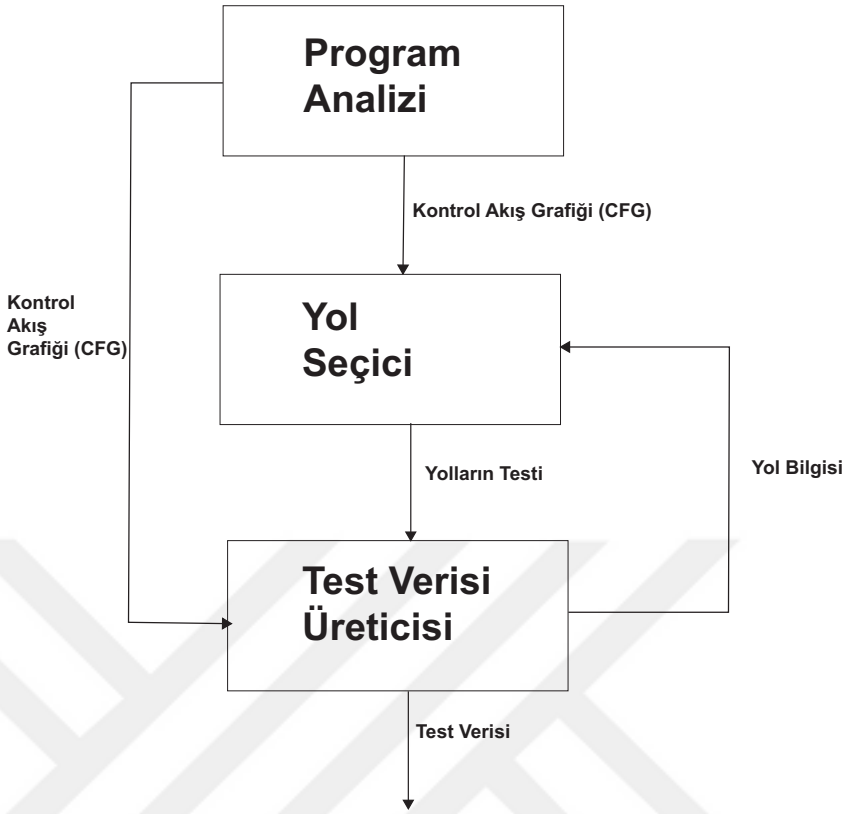
3.1. Kullanılan Platform ve Mimariler

Bu çalışmada Şekil 3.1'te görülen ve program analizi, yol seçicisi ve test verisi üreticisi parçalarından oluşan mimari kullanılmıştır. Program analizi almış olduğu kaynak kodu kontrol akış grafiğine çevirmektedir. Yol seçici kontrol akış grafiğinde gidilmesi gereken yola karar verir ve test verisi üreticisi ile bu bilgiyi paylaşmaktadır. Test verisi üreticisi ile çeşitli algoritmaları kullanarak ilgili yolu kapsayacak bir test verisi üretmektedir. Temel problemler grubunda triangle, even-odd, quadratic equation, largest number, leap year ve division on mark problemlerini içeren yedi farklı problem kullanılmıştır. Döngü içeren problemlerde ise asal sayılar, armstrong ve palindrom sayılar problemleri kullanılmıştır. Bütün problemler integer veri tipinde kodlanmıştır. Problemler Intel Xeon X5660 2.80 GHz işlemci ve 16 GB Ram üzerinde koşulmuştur. Tablo 3.2'de verilen problemler Python programlama dilinde kodlanmış olup meta-sezgisel algoritmalar ise MATLAB üzerinde çalıştırılmıştır.

Bu çalışmada amaç fonksiyonu minimizasyon problemi olarak ayarlanmıştır. Amaç fonksiyonu olarak appraximation level + branch distance, yol tabanlı amaç fonksiyonu ve benzerlik tabanlı amaç fonksiyonu kullanılmıştır.

Approximation level + branch distance (Eşitlik 3.1) amaç fonksiyonu branch distance (Eşitlik 3.2) ve approximation level metriklerinin birleşiminden oluşmaktadır.

$$fitness_{ALBD} = approximation\ level + normalize(branch\ distance) \tag{3.1}$$



Şekil 3.1. Çalışmada kullanılan test verisi üretimi mimarisi.

$$normalize(branch\ distance) = 1 - 1.001^{-branch\ distance} \tag{3.2}$$

Approximation level çalıştırılması gereken yol ile çalıştırılan yol incelenerek elde edilmektedir [65]. Bu metrik çalıştırılamayan yolların toplamından oluşmaktadır. Branch distance metriği ise Tracey'nin ortaya attığı [66] ve Tablo 3.1'da verilen yöntem ile hesaplanmaktadır. Bu değer ilgili dala girmek için ne kadar uzakta olduğumuzu belirtmektedir. Tablo 3.1'da görülen K değeri sonucun her zaman pozitif olmasını sağlayacak sabit bir değerdir.

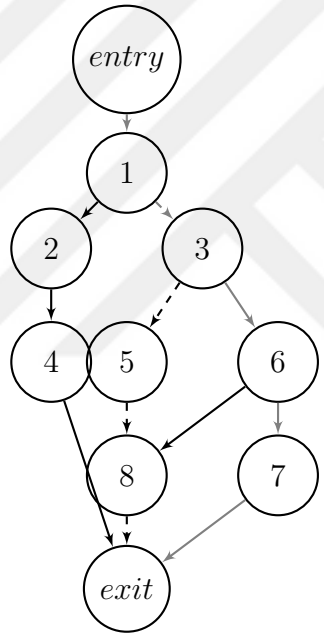
Tablo 3.1. Tracey'nin önermiş olduğu branch distance fonksiyonu.

Relational Predicate	Amaç Fonksiyonu
Boolean	if $TRUE$ then 0 else K
$a = b$	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 $b - a + K$

Yol tabanlı amaç fonksiyonu Eşitlik 3.3'da görüldüğü gibi hesaplanmaktadır:

$$fitness_{path\ based} = 1 - \frac{|\alpha \wedge \beta|}{|\alpha|} \tag{3.3}$$

α ve β hedef yol ve çalıştırılan yol düğüm setleridir, $|\alpha \wedge \beta|$ ise α ve β'nın doğru sıra ile eşleştiği düğüm sayısını vermektedir. Örneğin, Şekil 3.2'te verilen CFG için yol tabanlı amaç fonksiyonu $3/6 = 0.5$ olarak hesaplanmıştır. Çünkü hedef düğüm seti (α), $\{\mathrm{s}, 1, 3, 5, 8, \mathrm{e}\}$ düğümlerini içermektedir ve çalıştırılan yolların düğüm seti (β) $\{\mathrm{s}, 1, 3, 6, 7, \mathrm{e}\}$ düğümlerini içermektedir. Hedef düğüm setindeki $|\{\mathrm{s}, 1, 3, 5, 8, \mathrm{e}\}|$ ile doğru sırada eşleşen düğümler $|\{\mathrm{s}, 1, 3\}|$ olmaktadır.



Şekil 3.2. Yol tabanlı amaç fonksiyonu için CFG örneği.

Benzerlik tabanlı amaç fonksiyonu Eşitlik 3.4'da görüldüğü gibi hesaplanmaktadır:

$$fitness_{dissimilarity\ based} = \frac{|\alpha \oplus \beta|}{|\alpha \cup \beta|} \tag{3.4}$$

$|\alpha \oplus \beta|$ α ve β düğüm setlerinin simetrik farkının eleman sayısıdır. $|\alpha \cup \beta|$ ise α ve β düğüm setlerinin birleşiminin eleman sayısıdır. [67].

3.2. Temel Problemler Üzerinde Analizler

Bu bölümde Temel Problemler olarak kullanılan kod parçacıkların karakteristik özellikleri verilmiştir.

Tablo 3.2. Temel problemlerin karakteristikleri.

	Program	Satır Sayısı	Döngüsel Karmaşa	CFG Düğüm Sayısı	Değişken Sayısı
1	Triangle	23	9	23	3
2	Even Odd	6	2	5	1
3	Largest Number	11	4	8	3
4	Leap Year	7	4	8	1
5	Quadratic Equation	15	4	12	3
6	Remainder	7	3	6	2
7	Mark	19	11	22	3

- **Üçgen (Triangle)**: Bu program aldığı üç değerin üçgen oluşturup oluşturmadığına bakmaktadır. Eğer üçgen oluşturuyorsa da eşkenar, ikizkenar veya çeşitkenar üçgen olup olmadıklarına karar vermektedir.

- **Karesel Denklem (Quadratic Equation)**: Bu program aldığı üç değere göre $ax^2 + bx+c$ formatında olup olmadığına bakmakta, daha sonra diskriminant hesabını yaparak denklemin köklerini hesaplamaktadır.

- **Çift-Tek (Even-Odd)**: Bu program girilen değerin tek mi çift mi olduğuna karar vermektedir.

- **En Büyük Sayı (Largest Number)**: Bu program girilen üç değer arasında en büyük değeri bulmaktadır.

- **Kalan (Remainder)**: Bu program aldığı iki değere bakarak bölenin sıfır olup olmadığına bakarak kalan hesabı yapmaktadır.

- **Artık Yıl (Leap Year)**: Bu program girilen değerin artık yıl olup olmadığına bakmaktadır.

- **Not (Mark)**: Bu program aldığı üç değere göre ortalama hesaplamaktadır. Daha sonra ise elde ettiği ortalamaya göre beş farklı kategoride (A, B, C, D, E) kümelemektedir.

Problemlerin karakteristik özellikleri Tablo 3.2'da görülmektedir.

3.2.1. Parametre Analizi

Seçilen kontrol parametreleri algoritmaların performanslarını oldukça etkilemektedir. Doğru seçilecek parametreler sonucun etkin ve efektif olmasını sağlamaktadır. Bu yüzden algoritmaların parametre analizi önemli bir konudur. Bütün algoritmalar için popülasyon büyüklüğü 20 ve her bir yol için de 250 döngü sayısı sabit olarak seçilmiştir. Parametre analizi yaparken test edilen parametrelerin araştırılacağı uzayın alt ve üst sınır [-100,100] olarak belirlenmiştir.

ABC algoritması için popülasyon büyüklüğü ve iterasyon sayısı haricinde tek kontrol parametresi limit değeridir. Beş farklı limit değerinde algoritmanın performansı gözlenmiştir. PSO algoritması için inertia weight (w), social compenent (c_1) ve cognitive compenent (c_2) kontrol parametreleridir. Bu parametrelerin 252 farklı konfigürasyonunda PSO algoritması koşulmuştur. DE algoritması için crossover rate (CR) ve scaling factor (F) kontrol parametreleridir. 48 farklı kontrol parametresi için test edilmiştir. FA algoritması için α, β ve γ kontrol parametreleridir. Test edilen farklı kontrol parametresi ise 2000 tanedir. Kontrol parametrelerinin değerleri Tablo 3.3'da görülmektedir.

Tablo 3.3. Test edilen kontrol parametreleri, NP yiyecek kaynağı sayısı ve D problemin boyutunu belirtmektedir.

ABC	PSO	DE	FA
$limit$=$\{ NP*D*0.2$, $NP*D*0.4$, $NP*D*0.6$, $NP*D*0.8$, $NP*D\ \}$	$w = \{0.4, 0.5, 0.6, ..., 1\}$ $c_1 = \{1, 1.2, 1.4, ..., 2\}$ $c_2 = \{1, 1.2, 1.4, ..., 2\}$	$F = \{0.5, 0.6, 0.7, ..., 1\}$ $CR = \{0.3, 0.4, 0.5, ..., 1\}$	$\alpha = \{0.1, 0.2, 0.3, ..., 1\}$ $\beta_{min} = \{0.1, 0.2, 0.3, ..., 1\}$ $\gamma = \{0.5, 1, 1.5, ..., 10\}$

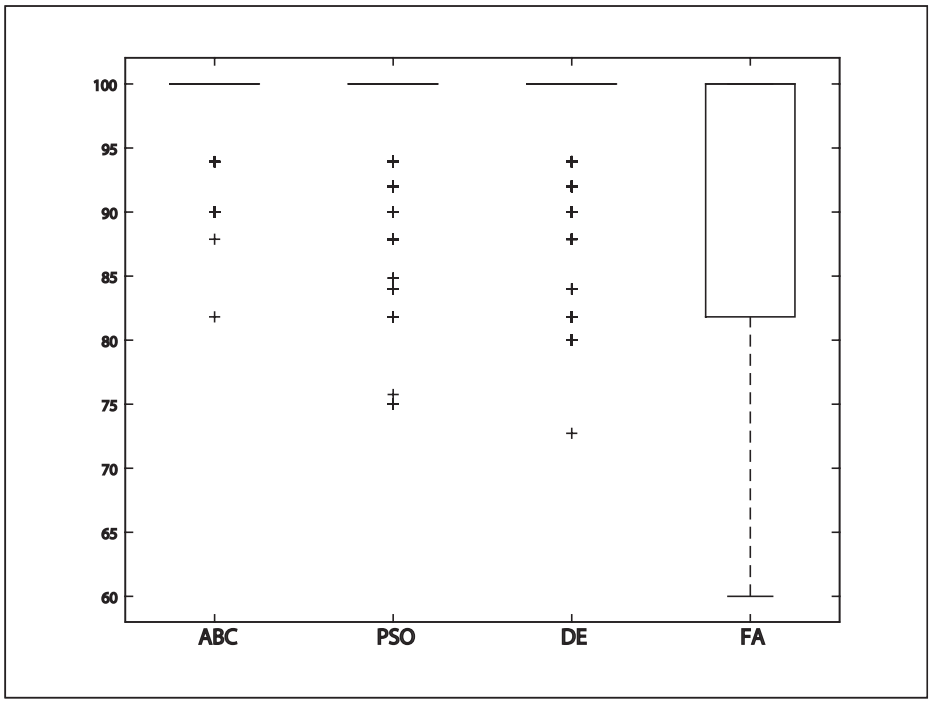
Her bir algoritma her bir konfigürasyon için 30 defa çalıştırılmıştır. Kapsama sonuçlarına göre önerilen parametre değerleri Tablo 3.4'te verilmiştir. Buna göre bütün sonuçların ortalama, standart sapma ve medyan değerleri ise Tablo 3.5'te verilmiştir. İlk satırda ortalama ve standart sapma değerleri, ikinci satırda ise medyan değeri verilmektedir. Şekil 3.3'da görüldüğü üzere en az standart sapmaya sahip algoritma ABC algoritmasıdır. Kontrol parametresinin değerinden en az etkilenen algoritma olduğunu göstermektedir.

Tablo 3.4. Temel problemlerde kullanılan her bir algoritma için önerilen kontrol parametreleri.

ABC	PSO	DE	FA
$Limit$: $NP * D$	w: 0.5 c_1: 1 c_2: 1.4	CR: 0.5 F: 1	α: 0.4 β_{min}: 0.7 γ: 6

Tablo 3.5. Temel problemler için farklı ayarlarda koşturulan algoritmaların istatistiksel sonuçları. Ortalama $\mp$ Standart Sapma and Medyan değerleri verilmiştir.

Program	ABC	PSO	DE	FA
Triangle	**99.3535 $\mp$ 1.1193** (**99.7980**)	98.4873 $\mp$ 1.2754 (98.7879)	99.6086 $\mp$ 0.4197 (99.5960)	99.3798 $\mp$ 0.5221 (99.3939)
Even Odd	**100.0000 $\mp$ 0.0000** (**100.0000**)	99.9537 $\mp$ 0.2052 (100.0000)	**100.0000 $\mp$ 0.0000** (**100.0000**)	**100.0000 $\mp$ 0.0000** (**100.0000**)
Largest	**100.0000 $\mp$ 0.0000** (**100.0000**)	**100.0000 $\mp$ 0.0000** (**100.0000**)	**100.0000 $\mp$ 0.0000** (**100.0000**)	99.9996 $\mp$ 0.0141 (100.0000)
Leap Year	**100.0000 $\mp$ 0.0000** (**100.0000**)	94.8341 $\mp$ 3.0361 (95.1515)	88.8258 $\mp$ 1.8455 (88.4848)	88.7103 $\mp$ 2.2053 (88.4848)
Quadratic Equation	96.8667 $\mp$ 0.6055 (96.6667)	**97.7143 $\mp$ 1.6428** (**97.6667**)	90.2222 $\mp$ 3.7393 (91.0000)	81.0642 $\mp$ 0.5613 (81.0000)
Remainder	**100.0000 $\mp$ 0.0000** (**100.0000**)	**100.0000 $\mp$ 0.0000** (**100.0000**)	99.9811 $\mp$ 0.1312 (100.0000)	90.8586 $\mp$ 2.3459 (90.9091)
Mark	**100.0000 $\mp$ 0.0000** (**100.0000**)	99.4508 $\mp$ 0.4370 (99.4667)	99.5556 $\mp$ 0.6034 (99.7333)	99.3845 $\mp$ 1.2059 (100.0000)



Şekil 3.3. Temel problemler için farklı ayarlarda koşturulan algoritmaların boxplot grafikleri.

3.2.2. Meta-sezgisel Algoritmaların Kıyaslanması

Meta-sezgisel algoritmaların kıyaslanması için [-100,100] aralığını içeren arama uzayı ile [-1000,1000] aralığını içeren iki farklı arama uzayı seçilmiştir. Her iki uzaya ait sonuçlar Tablo 3.6,3.7 ve 3.8'da verilmektedir. Tabloların ilk satırında kapsama miktarı, döngü sayısı ve çalışma zamanının ortalama ve standart sapması, ikinci satırında ise medyan değerler verilmektedir. Ayrıca koyu renkte algoritmanın kaçıncı olduğu belirtilmektedir. Yol tabanlı amaç fonksiyonu kullanılması durumunda elde edilen sonuçlar (Tablo 3.6), benzerlik tabanlı amaç fonksiyonu kullanılması durumunda elde edilen sonuçlar (Tablo 3.7) ve approximation level + branch distance amaç fonksiyonu kullanılması durumunda elde edilen sonuçlar (Tablo 3.8)'de sunulmaktadır.

3.2.2.1. Yol Tabanlı Amaç Fonksiyonu Kıyaslaması

Yol tabanlı amaç fonksiyonu kullanılması durumunda elde edilen sonuçlar Tablo 3.6'da verilmiştir. Tablo 3.6'daki verilere göre; [-100,100] aralığında ortalama ve standart sapma değerlerine göre incelendiğinde ABC ve PSO algoritmaları benzer performanslar göstermiştir. ABC algoritması triangle ve leap problemlerinde daha iyiyken PSO algoritması quadratic probleminde DE algoritması ise mark probleminde daha iyi sonuçlar üretmiştir. Parantez içerisinde gösterilen medyan sonuçları incelendiğinde ise ABC algoritması triangle ve quadratic problemleri hariç diğer problemlerde %100 kapsama miktarına erişmiştir. PSO algoritması ise triangle problemi hariç diğer problemlerde %100 kapsama miktarına erişmiştir. DE algoritması ise leap ve quadratic hariç diğer problemlerde %100 kapsama miktarını elde etmiştir. Döngü sayısı ve çalışma zamanları incelendiğinde optimuma erişebilen algoritmaların kısa sürede sonuca eriştiği görülmektedir. Çalışma zamanı maksimum döngü sayısı ile de ilişkili olduğu için bütün problemlerde aynı döngüde çalışmayı bitiren even-odd problemi incelenmelidir. Even-odd problemi incelendiğinde FA algoritmasının diğer algoritmalardan daha hızlı olduğu görülmektedir.

Arama uzayı [-1000,1000] olduğunda algoritmaların sonuçlarında gerileme olduğu görülmektedir. Bunun nedeni leap problemi hariç problemlerin daha zorlaşmasıdır.

Tablo 3.6. Temel problemler için algoritmaların yol tabanlı amaç fonksiyonu kullanarak elde ettikleri sonuçlar ortalama ∓ standart sapma olarak verilmiştir. Parantez içinde medyan değer, koyu renkte ise algoritmanın diğer algoritmalara kıyasla sırası verilmektedir. BS başarı sırasını ifade etmektedir.

		Aralık							
	Program	[-100,100]				[-1000,1000]			
		Kapsama	BS	Çevrim	Zaman	Kapsama	BS	Çevrim	Zaman
ABC	Triangle	96.3636 ∓ 3.0198 (93.9394)	**2**	307.5667 ∓ 122.1601 (310.0000)	2.2112 ∓ 0.8496 (2.1702)	88.6869 ∓ 7.9553 (93.9394)	**2**	452.8333 ∓ 87.5947 (502.0000)	3.4718 ∓ 0.6637 (3.7565)
	Even Odd	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0474 ∓ 0.0521 (0.0365)	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0476 ∓ 0.0390 (0.0400)
	Largest	100.0000 ∓ 0.0000 (100.0000)	**1**	4.3000 ∓ 0.9523 (4.0000)	0.0685 ∓ 0.0709 (0.0534)	100.0000 ∓ 0.0000 (100.0000)	**1**	4.1000 ∓ 0.4026 (4.0000)	0.0603 ∓ 0.0347 (0.0532)
	Leap	100.0000 ∓ 0.0000 (100.0000)	**1**	19.5333 ∓ 16.1068 (13.5000)	0.1468 ∓ 0.1047 (0.1084)	100.0000 ∓ 0.0000 (100.0000)	**1**	24.9667 ∓ 19.3684 (21.0000)	0.2094 ∓ 0.1453 (0.1770)
	Quadratic	93.6667 ∓ 4.9013 (90.0000)	**2**	230.8000 ∓ 95.6273 (261.0000)	1.5530 ∓ 0.6324 (1.7592)	85.6667 ∓ 5.0401 (90.0000)	**2**	411.9000 ∓ 93.7231 (436.5000)	3.1526 ∓ 0.6870 (3.3642)
	Remainder	100.0000 ∓ 0.0000 (100.0000)	**1**	2.5333 ∓ 1.2521 (2.0000)	0.0359 ∓ 0.0189 (0.0281)	100.0000 ∓ 0.0000 (100.0000)	**1**	10.5667 ∓ 9.9123 (8.0000)	0.1048 ∓ 0.1083 (0.0715)
	Mark	99.7333 ∓ 1.4606 (100.0000)	**2**	39.0000 ∓ 43.8807 (30.0000)	0.3410 ∓ 0.3324 (0.2737)	98.6667 ∓ 3.0324 (100.0000)	**1**	310.8000 ∓ 87.6806 (301.5000)	2.4353 ∓ 0.7186 (2.3606)
PSO	Triangle	84.7475 ∓ 8.2504 (78.7879)	**3**	445.7333 ∓ 95.8306 (502.0000)	3.1110 ∓ 0.6566 (3.4768)	82.8283 ∓ 6.2829 (80.3030)	**3**	489.6333 ∓ 42.9575 (502.0000)	3.7422 ∓ 0.3194 (3.8215)
	Even Odd	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0297 ∓ 0.0110 (0.0267)	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0268 ∓ 0.0037 (0.0259)
	Largest	100.0000 ∓ 0.0000 (100.0000)	**1**	4.0000 ∓ 0.0000 (4.0000)	0.0597 ∓ 0.0040 (0.0586)	100.0000 ∓ 0.0000 (100.0000)	**1**	4.0000 ∓ 0.0000 (4.0000)	0.0560 ∓ 0.0061 (0.0546)
	Leap	99.3939 ∓ 3.3195 (100.0000)	**2**	23.0667 ∓ 47.6032 (7.0000)	0.1616 ∓ 0.2931 (0.0632)	100.0000 ∓ 0.0000 (100.0000)	**1**	21.7000 ∓ 25.5992 (11.5000)	0.1669 ∓ 0.1705 (0.0989)
	Quadratic	97.3333 ∓ 4.4978 (100.0000)	**1**	138.8333 ∓ 98.9044 (131.5000)	0.9346 ∓ 0.6350 (0.8671)	90.0000 ∓ 0.0000 (90.0000)	**1**	293.2667 ∓ 46.6113 (275.5000)	2.1890 ∓ 0.3218 (2.0892)
	Remainder	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0667 ∓ 0.2537 (2.0000)	0.0299 ∓ 0.0035 (0.0289)	100.0000 ∓ 0.0000 (100.0000)	**1**	6.3667 ∓ 4.5674 (6.0000)	0.0597 ∓ 0.0298 (0.0576)
	Mark	97.8667 ∓ 3.5982 (100.0000)	**3**	88.3000 ∓ 130.9304 (13.0000)	0.7524 ∓ 1.0372 (0.1493)	89.8667 ∓ 8.8969 (92.0000)	**3**	503.9333 ∓ 181.8502 (479.0000)	3.7925 ∓ 1.3504 (3.6017)
DE	Triangle	95.8586 ∓ 5.3076 (96.9697)	**1**	258.3333 ∓ 145.1780 (291.0000)	2.1507 ∓ 1.1738 (2.4155)	89.4949 ∓ 7.7046 (93.9394)	**1**	426.2333 ∓ 113.8149 (502.0000)	3.4288 ∓ 0.9130 (3.9590)
	Even Odd	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0209 ∓ 0.0033 (0.0199)	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0316 ∓ 0.0522 (0.0215)
	Largest	100.0000 ∓ 0.0000 (100.0000)	**1**	4.0000 ∓ 0.0000 (4.0000)	0.0452 ∓ 0.0051 (0.0440)	100.0000 ∓ 0.0000 (100.0000)	**1**	4.1000 ∓ 0.3051 (4.0000)	0.0533 ∓ 0.0234 (0.0457)
	Leap	89.6970 ∓ 9.1638 (81.8182)	**3**	146.0333 ∓ 123.2946 (252.0000)	1.1036 ∓ 0.9186 (1.8577)	97.5758 ∓ 6.2863 (100.0000)	**2**	52.2000 ∓ 85.9304 (18.0000)	0.3956 ∓ 0.6214 (0.1477)
	Quadratic	93.6667 ∓ 6.1495 (90.0000)	**3**	240.2333 ∓ 104.1768 (254.5000)	1.8464 ∓ 0.7690 (1.9565)	82.6667 ∓ 4.4978 (80.0000)	**3**	465.5000 ∓ 68.8285 (502.0000)	3.7032 ∓ 0.5596 (3.9326)
	Remainder	100.0000 ∓ 0.0000 (100.0000)	**1**	2.6000 ∓ 1.1626 (2.0000)	0.0316 ∓ 0.0128 (0.0244)	100.0000 ∓ 0.0000 (100.0000)	**1**	10.7000 ∓ 11.5763 (6.0000)	0.0970 ∓ 0.0865 (0.0614)
	Mark	100.0000 ∓ 0.0000 (100.0000)	**1**	9.0667 ∓ 1.7991 (9.0000)	0.1298 ∓ 0.0224 (0.1321)	97.3333 ∓ 4.3734 (100.0000)	**2**	332.1000 ∓ 134.1706 (283.5000)	2.6609 ∓ 1.0489 (2.3147)
FA	Triangle	81.3131 ∓ 4.5076 (78.7879)	**4**	493.7667 ∓ 45.2851 (502.0000)	3.4757 ∓ 0.3244 (3.5157)	81.4141 ∓ 3.4444 (81.8182)	**4**	510.3333 ∓ 45.4550 (502.0000)	3.4751 ∓ 0.2902 (3.4225)
	Even Odd	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0188 ∓ 0.0034 (0.0178)	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0196 ∓ 0.0031 (0.0188)
	Largest	98.2222 ∓ 4.6099 (100.0000)	**2**	37.2000 ∓ 86.0907 (4.0000)	0.2296 ∓ 0.4882 (0.0399)	99.5556 ∓ 2.4343 (100.0000)	**2**	12.3000 ∓ 45.4610 (4.0000)	0.0912 ∓ 0.2620 (0.0419)
	Leap	89.6970 ∓ 9.1638 (81.8182)	**3**	145.1667 ∓ 124.3319 (252.0000)	0.8698 ∓ 0.7355 (1.4465)	86.6667 ∓ 8.1778 (81.8182)	**3**	195.8667 ∓ 122.4724 (252.0000)	1.1503 ∓ 0.7080 (1.4711)
	Quadratic	81.3333 ∓ 3.4575 (80.0000)	**4**	468.8000 ∓ 86.0907 (502.0000)	3.0160 ∓ 0.5418 (3.2203)	80.0000 ∓ 0.0000 (80.0000)	**4**	502.0000 ∓ 0.0000 (502.0000)	3.2823 ∓ 0.0400 (3.2780)
	Remainder	91.8182 ∓ 12.7116 (100.0000)	**2**	76.7000 ∓ 116.0568 (2.0000)	0.4475 ∓ 0.6627 (0.0215)	75.4545 ∓ 8.3217 (72.7273)	**2**	226.1000 ∓ 75.9770 (251.0000)	1.3090 ∓ 0.4357 (1.4511)
	Mark	95.4667 ∓ 5.4313 (100.0000)	**4**	163.1000 ∓ 190.3377 (22.0000)	1.2404 ∓ 1.3989 (0.1958)	39.2000 ∓ 23.9171 (20.0000)	**4**	1120.2000 ∓ 176.2470 (1251.0000)	7.5072 ∓ 1.1657 (8.1706)

Tablo 3.7. Temel problemler için algoritmaların dissimilarity tabanlı amaç fonksiyonu kullanarak elde ettiği sonuçlar ortalama ∓ standart sapma olarak verilmiştir. Parantez içinde medyan değer, koyu renkte ise algoritmanın diğer algoritmalara kıyasla sırası verilmektedir. BS başarı sırasını ifade etmektedir.

		Aralık							
	Program	[-100,100]				[-1000,1000]			
		Kapsama	BS	Çevrim	Zaman	Kapsama	BS	Çevrim	Zaman
ABC	Triangle	96.0606 ∓ 3.0975 (93.9394)	**2**	299.7667 ∓ 136.6777 (309.0000)	2.3190 ∓ 1.0376 (2.3551)	91.8182 ∓ 3.0975 (93.9394)	**2**	452.4000 ∓ 84.0889 (502.0000)	3.7787 ∓ 0.7194 (4.1480)
	Even Odd	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0472 ∓ 0.0447 (0.0358)	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0489 ∓ 0.0341 (0.0406)
	Largest	100.0000 ∓ 0.0000 (100.0000)	**1**	4.1667 ∓ 0.5307 (4.0000)	0.0774 ∓ 0.0734 (0.0604)	100.0000 ∓ 0.0000 (100.0000)	**1**	4.3667 ∓ 0.8899 (4.0000)	0.0680 ∓ 0.0366 (0.0608)
	Leap	100.0000 ∓ 0.0000 (100.0000)	**1**	16.9667 ∓ 22.2067 (6.0000)	0.1361 ∓ 0.1496 (0.0574)	100.0000 ∓ 0.0000 (100.0000)	**1**	28.1667 ∓ 28.3793 (16.5000)	0.2463 ∓ 0.2188 (0.1501)
	Quadratic	97.3333 ∓ 4.4978 (100.0000)	**1**	185.2667 ∓ 87.3909 (198.5000)	1.3513 ∓ 0.6255 (1.4708)	87.0000 ∓ 4.6609 (90.0000)	**2**	401.0000 ∓ 100.6153 (429.5000)	3.1629 ∓ 0.8059 (3.3703)
	Remainder	100.0000 ∓ 0.0000 (100.0000)	**1**	2.5333 ∓ 1.2521 (2.0000)	0.0325 ∓ 0.0210 (0.0254)	100.0000 ∓ 0.0000 (100.0000)	**1**	10.5667 ∓ 9.9123 (8.0000)	0.1010 ∓ 0.1066 (0.0670)
	Mark	100.0000 ∓ 0.0000 (100.0000)	**1**	36.7000 ∓ 21.0781 (29.5000)	0.3448 ∓ 0.1684 (0.2850)	98.6667 ∓ 3.0324 (100.0000)	**2**	294.0333 ∓ 86.6151 (273.0000)	2.4862 ∓ 0.7367 (2.2679)
PSO	Triangle	89.5960 ∓ 7.3232 (93.9394)	**3**	413.0333 ∓ 115.4035 (502.0000)	3.1002 ∓ 0.8574 (3.6443)	82.9293 ∓ 6.2399 (78.7879)	**3**	494.2333 ∓ 42.9191 (502.0000)	3.9259 ∓ 0.3482 (3.9751)
	Even Odd	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0283 ∓ 0.0049 (0.0265)	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0272 ∓ 0.0039 (0.0263)
	Largest	100.0000 ∓ 0.0000 (100.0000)	**1**	4.0000 ∓ 0.0000 (4.0000)	0.0657 ∓ 0.0084 (0.0611)	100.0000 ∓ 0.0000 (100.0000)	**1**	4.0000 ∓ 0.0000 (4.0000)	0.0565 ∓ 0.0050 (0.0547)
	Leap	98.7879 ∓ 4.6129 (100.0000)	**2**	29.8333 ∓ 61.7967 (10.0000)	0.2057 ∓ 0.3833 (0.0878)	99.3939 ∓ 3.3195 (100.0000)	**2**	31.1333 ∓ 45.4599 (19.5000)	0.2438 ∓ 0.3275 (0.1593)
	Quadratic	97.3333 ∓ 4.4978 (100.0000)	**1**	149.2333 ∓ 90.2329 (123.5000)	1.0749 ∓ 0.6291 (0.8714)	89.3333 ∓ 2.5371 (90.0000)	**1**	321.0000 ∓ 70.0118 (295.5000)	2.4781 ∓ 0.5381 (2.2783)
	Remainder	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0667 ∓ 0.2537 (2.0000)	0.0277 ∓ 0.0037 (0.0267)	100.0000 ∓ 0.0000 (100.0000)	**1**	6.3667 ∓ 4.5674 (6.0000)	0.0584 ∓ 0.0303 (0.0583)
	Mark	99.4667 ∓ 2.0297 (100.0000)	**2**	32.7333 ∓ 63.7489 (11.0000)	0.3212 ∓ 0.5285 (0.1409)	90.1333 ∓ 9.0810 (92.0000)	**3**	485.2000 ∓ 234.5670 (482.5000)	3.9885 ∓ 1.9202 (3.9995)
DE	Triangle	95.7576 ∓ 5.6046 (100.0000)	**1**	274.6000 ∓ 148.6413 (279.0000)	2.4146 ∓ 1.2845 (2.4398)	92.0202 ∓ 5.8198 (93.9394)	**1**	413.7667 ∓ 87.5894 (423.5000)	3.6015 ∓ 0.7467 (3.7321)
	Even Odd	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0212 ∓ 0.0034 (0.0201)	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0235 ∓ 0.0039 (0.0228)
	Largest	100.0000 ∓ 0.0000 (100.0000)	**1**	4.0000 ∓ 0.0000 (4.0000)	0.0487 ∓ 0.0075 (0.0457)	100.0000 ∓ 0.0000 (100.0000)	**1**	4.1333 ∓ 0.7303 (4.0000)	0.0507 ∓ 0.0103 (0.0470)
	Leap	90.3030 ∓ 9.2258 (81.8182)	**3**	138.2000 ∓ 123.7546 (252.0000)	1.0657 ∓ 0.9422 (1.8965)	93.9394 ∓ 8.7175 (100.0000)	**3**	100.9667 ∓ 110.4428 (34.0000)	0.7934 ∓ 0.8462 (0.2844)
	Quadratic	93.6667 ∓ 6.1495 (90.0000)	**2**	256.7333 ∓ 135.2838 (255.5000)	2.1311 ∓ 1.1090 (2.1264)	83.3333 ∓ 5.4667 (80.0000)	**3**	451.6000 ∓ 85.2577 (502.0000)	3.7672 ∓ 0.7085 (4.1273)
	Remainder	100.0000 ∓ 0.0000 (100.0000)	**1**	2.6000 ∓ 1.1626 (2.0000)	0.0317 ∓ 0.0135 (0.0261)	100.0000 ∓ 0.0000 (100.0000)	**1**	10.7000 ∓ 11.5763 (6.0000)	0.0928 ∓ 0.0848 (0.0607)
	Mark	100.0000 ∓ 0.0000 (100.0000)	**1**	10.5000 ∓ 2.1456 (10.0000)	0.1542 ∓ 0.0256 (0.1476)	98.9333 ∓ 2.7660 (100.0000)	**1**	276.5333 ∓ 97.0534 (247.0000)	2.4408 ∓ 0.8446 (2.1807)
FA	Triangle	81.6162 ∓ 5.5668 (78.7879)	**4**	486.8667 ∓ 82.8887 (502.0000)	3.6874 ∓ 0.6194 (3.8006)	79.4949 ∓ 4.0330 (78.7879)	**4**	510.5000 ∓ 45.6144 (502.0000)	3.7817 ∓ 0.3452 (3.7214)
	Even Odd	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0202 ∓ 0.0044 (0.0185)	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0215 ∓ 0.0072 (0.0193)
	Largest	98.6667 ∓ 4.0684 (100.0000)	**2**	28.9000 ∓ 75.9770 (4.0000)	0.1960 ∓ 0.4658 (0.0422)	99.5556 ∓ 2.4343 (100.0000)	**2**	12.3333 ∓ 45.4550 (4.0000)	0.0931 ∓ 0.2780 (0.0413)
	Leap	85.4545 ∓ 7.3971 (81.8182)	**4**	202.4333 ∓ 100.8289 (252.0000)	1.2414 ∓ 0.6142 (1.5200)	89.0909 ∓ 9.0595 (81.8182)	**4**	154.8333 ∓ 121.2023 (252.0000)	0.9783 ∓ 0.7569 (1.5543)
	Quadratic	81.3333 ∓ 3.4575 (80.0000)	**3**	468.8333 ∓ 86.0044 (502.0000)	3.2869 ∓ 0.6071 (3.5154)	80.3333 ∓ 1.8257 (80.0000)	**4**	493.7000 ∓ 45.4610 (502.0000)	3.4319 ∓ 0.3030 (3.4798)
	Remainder	91.8182 ∓ 12.7116 (100.0000)	**2**	76.7000 ∓ 116.0568 (2.0000)	0.4679 ∓ 0.6975 (0.0195)	75.4545 ∓ 8.3217 (72.7273)	**2**	226.1000 ∓ 75.9770 (251.0000)	1.3550 ∓ 0.4516 (1.4998)
	Mark	97.0667 ∓ 3.9211 (100.0000)	**3**	113.2667 ∓ 138.6954 (17.0000)	0.9412 ∓ 1.1301 (0.1696)	40.8000 ∓ 23.8246 (20.0000)	**4**	1093.8333 ∓ 195.1159 (1251.0000)	8.1136 ∓ 1.4587 (9.1300)

Tablo 3.8. Temel problemler için algoritmaların approximation level + branch distance amaç fonksiyonu kullanarak elde ettiği sonuçlar ortalama ∓ standart sapma olarak verilmiştir. Parantez içinde medyan değer, koyu renkte ise algoritmanın diğer algoritmalara kıyasla sırası verilmektedir. BS başarı sırasını ifade etmektedir.

		Aralık							
	Program	[-100,100]				[-1000,1000]			
		Kapsama	BS	Çevrim	Zaman	Kapsama	BS	Çevrim	Zaman
ABC	Triangle	100.0000 ∓ 0.0000 (100.0000)	**1**	83.4000 ∓ 43.4532 (78.5000)	0.5722 ∓ 0.2878 (0.5380)	95.4545 ∓ 3.4459 (93.9394)	**3**	310.8667 ∓ 90.3215 (300.0000)	2.1778 ∓ 0.5914 (2.1717)
	Even Odd	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0353 ∓ 0.0019 (0.0345)	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.1019 ∓ 0.2212 (0.0367)
	Largest	100.0000 ∓ 0.0000 (100.0000)	**1**	4.1333 ∓ 0.4342 (4.0000)	0.0545 ∓ 0.0027 (0.0541)	100.0000 ∓ 0.0000 (100.0000)	**1**	4.0667 ∓ 0.2537 (4.0000)	0.0775 ∓ 0.0749 (0.0599)
	Leap	100.0000 ∓ 0.0000 (100.0000)	**1**	22.0000 ∓ 24.9634 (10.0000)	0.1755 ∓ 0.1810 (0.0857)	100.0000 ∓ 0.0000 (100.0000)	**1**	47.3333 ∓ 52.3446 (32.0000)	0.3555 ∓ 0.3545 (0.2460)
	Quadratic	96.3333 ∓ 4.9013 (100.0000)	**1**	195.3667 ∓ 99.7557 (211.0000)	1.3690 ∓ 0.6855 (1.4726)	84.3333 ∓ 5.0401 (80.0000)	**3**	421.4000 ∓ 102.9853 (502.0000)	3.2961 ∓ 0.8253 (3.8184)
	Remainder	100.0000 ∓ 0.0000 (100.0000)	**1**	2.1333 ∓ 0.4342 (2.0000)	0.0286 ∓ 0.0052 (0.0267)	100.0000 ∓ 0.0000 (100.0000)	**1**	10.2333 ∓ 9.3207 (8.0000)	0.1021 ∓ 0.1066 (0.0689)
	Mark	100.0000 ∓ 0.0000 (100.0000)	**1**	17.5333 ∓ 8.1652 (16.0000)	0.1806 ∓ 0.0627 (0.1703)	95.4667 ∓ 5.4313 (100.0000)	**1**	597.1333 ∓ 178.3640 (591.0000)	4.8220 ∓ 1.4271 (4.7468)
PSO	Triangle	100.0000 ∓ 0.0000 (100.0000)	**1**	28.6333 ∓ 18.5853 (26.5000)	0.2714 ∓ 0.1450 (0.2521)	99.5960 ∓ 1.5376 (100.0000)	**1**	104.3000 ∓ 79.7605 (69.0000)	0.8087 ∓ 0.5844 (0.5432)
	Even Odd	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0275 ∓ 0.0014 (0.0272)	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0281 ∓ 0.0027 (0.0274)
	Largest	100.0000 ∓ 0.0000 (100.0000)	**1**	4.0000 ∓ 0.0000 (4.0000)	0.0631 ∓ 0.0081 (0.0609)	100.0000 ∓ 0.0000 (100.0000)	**1**	4.0000 ∓ 0.0000 (4.0000)	0.0620 ∓ 0.0038 (0.0603)
	Leap	98.7879 ∓ 4.6129 (100.0000)	**2**	24.0667 ∓ 62.1927 (6.0000)	0.1949 ∓ 0.4569 (0.0633)	99.3939 ∓ 3.3195 (100.0000)	**2**	28.2333 ∓ 53.5013 (12.5000)	0.2182 ∓ 0.3642 (0.1116)
	Quadratic	95.6667 ∓ 5.0401 (100.0000)	**2**	162.5333 ∓ 98.9914 (178.0000)	1.3187 ∓ 0.7747 (1.4769)	90.0000 ∓ 0.0000 (90.0000)	**1**	279.3000 ∓ 23.5828 (273.5000)	2.1019 ∓ 0.2036 (2.0879)
	Remainder	100.0000 ∓ 0.0000 (100.0000)	**1**	2.1000 ∓ 0.3051 (2.0000)	0.0354 ∓ 0.0068 (0.0334)	100.0000 ∓ 0.0000 (100.0000)	**1**	5.7333 ∓ 6.5175 (3.5000)	0.0555 ∓ 0.0425 (0.0422)
	Mark	99.7333 ∓ 1.4606 (100.0000)	**2**	18.1667 ∓ 45.3249 (10.0000)	0.2220 ∓ 0.3636 (0.1526)	35.4667 ∓ 21.4182 (20.0000)	**2**	1137.2000 ∓ 169.6073 (1251.0000)	8.9363 ∓ 1.3621 (9.7399)
DE	Triangle	99.5960 ∓ 1.5376 (100.0000)	**2**	64.0000 ∓ 61.8942 (45.5000)	0.6198 ∓ 0.5835 (0.4352)	99.3939 ∓ 1.8493 (100.0000)	**2**	132.9667 ∓ 98.3872 (105.5000)	1.0712 ∓ 0.7790 (0.8795)
	Even Odd	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0232 ∓ 0.0031 (0.0224)	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0265 ∓ 0.0144 (0.0222)
	Largest	100.0000 ∓ 0.0000 (100.0000)	**1**	4.0000 ∓ 0.0000 (4.0000)	0.0523 ∓ 0.0091 (0.0502)	100.0000 ∓ 0.0000 (100.0000)	**1**	4.0000 ∓ 0.0000 (4.0000)	0.0488 ∓ 0.0090 (0.0456)
	Leap	88.4848 ∓ 8.9115 (81.8182)	**3**	161.9333 ∓ 120.4165 (252.0000)	1.3474 ∓ 0.9887 (2.0559)	95.7576 ∓ 7.8215 (100.0000)	**3**	70.8667 ∓ 101.9573 (20.5000)	0.5437 ∓ 0.7573 (0.1614)
	Quadratic	95.3333 ∓ 5.0742 (100.0000)	**3**	218.5333 ∓ 96.4142 (255.0000)	1.9180 ∓ 0.8245 (2.2291)	86.0000 ∓ 4.9827 (90.0000)	**2**	409.4333 ∓ 93.1346 (415.5000)	3.2442 ∓ 0.7118 (3.2622)
	Remainder	100.0000 ∓ 0.0000 (100.0000)	**1**	2.6667 ∓ 1.4933 (2.0000)	0.0337 ∓ 0.0146 (0.0275)	100.0000 ∓ 0.0000 (100.0000)	**1**	13.2333 ∓ 12.2719 (10.5000)	0.1145 ∓ 0.0950 (0.1003)
	Mark	100.0000 ∓ 0.0000 (100.0000)	**1**	9.3333 ∓ 1.6470 (9.0000)	0.1426 ∓ 0.0204 (0.1410)	32.8000 ∓ 20.9670 (20.0000)	**4**	1161.3667 ∓ 149.8734 (1251.0000)	9.6179 ∓ 1.2815 (10.0871)
FA	Triangle	99.3939 ∓ 2.4399 (100.0000)	**3**	124.2333 ∓ 39.6942 (116.0000)	0.8316 ∓ 0.2494 (0.7764)	94.8485 ∓ 2.4072 (93.9394)	**4**	345.1667 ∓ 94.8440 (324.5000)	2.6072 ∓ 0.6946 (2.4857)
	Even Odd	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0177 ∓ 0.0005 (0.0177)	100.0000 ∓ 0.0000 (100.0000)	**1**	2.0000 ∓ 0.0000 (2.0000)	0.0196 ∓ 0.0031 (0.0188)
	Largest	100.0000 ∓ 0.0000 (100.0000)	**1**	4.1667 ∓ 0.5307 (4.0000)	0.0434 ∓ 0.0056 (0.0410)	100.0000 ∓ 0.0000 (100.0000)	**1**	4.0333 ∓ 0.1826 (4.0000)	0.0449 ∓ 0.0068 (0.0423)
	Leap	87.2727 ∓ 8.4744 (81.8182)	**4**	178.5667 ∓ 114.1287 (252.0000)	1.0838 ∓ 0.6862 (1.5002)	89.6970 ∓ 9.1638 (81.8182)	**4**	147.9333 ∓ 121.4825 (252.0000)	0.8547 ∓ 0.6926 (1.4186)
	Quadratic	82.3333 ∓ 4.3018 (80.0000)	**4**	443.9000 ∓ 107.1156 (502.0000)	2.8577 ∓ 0.6746 (3.2168)	80.0000 ∓ 0.0000 (80.0000)	**4**	502.0000 ∓ 0.0000 (502.0000)	3.2695 ∓ 0.1039 (3.2440)
	Remainder	91.8182 ∓ 12.7116 (100.0000)	**2**	76.7000 ∓ 116.0568 (2.0000)	0.4462 ∓ 0.6624 (0.0211)	79.0909 ∓ 11.7323 (72.7273)	**2**	192.9000 ∓ 107.1156 (251.0000)	1.1529 ∓ 0.6338 (1.4845)
	Mark	99.7333 ∓ 1.4606 (100.0000)	**2**	28.8000 ∓ 46.4122 (21.0000)	0.2308 ∓ 0.3163 (0.1773)	35.4667 ∓ 22.0278 (20.0000)	**3**	1156.1333 ∓ 158.1840 (1251.0000)	8.2334 ∓ 1.1089 (8.8328)

Leap probleminde araştırma uzayındaki optimum çözüm sayısı arttığı için problem nispeten kolaylaşmıştır. Even-odd, largest ve remainder problemlerinde ABC, DE ve PSO algoritmaları benzer sonuçlar üretmiştir. ABC algoritması mark probleminde, DE algoritması triangle probleminde ve PSO algoritması da quadratic probleminde en iyi sonucu üreten algoritmalar olmuştur. Algoritmaların çalışma zamanlarında artışlar gözlenmiştir. Optimuma eriştiğinde sonlanan algoritmalar arama uzayı büyüdüğünde daha geç optimuma erişmiş veya erişememiştir. Çalışma zamanı en düşük algoritma FA olsa da en iyi sonucu üretemeyip daha fazla iterasyonda çalışması gerektiği için yakınsama hızı açısından yavaş kalmıştır.

3.2.2.2. Benzerlik Tabanlı Amaç Fonksiyonu Kıyaslaması

Benzerlik tabanlı amaç fonksiyonu kullanılması durumunda elde edilen sonuçlar Tablo 3.7'de verilmektedir. Tablo 3.7'deki ortalama sonuçlarına göre ABC algoritması arama uzayı [-100,100] aralığında iken en iyi sonuçları üretmiştir. Arama uzayı [-1000,1000] olduğunda ise even-odd ve largest problemlerinde algoritmaların benzer sonuçlar ürettiği gözlemlenmiştir. Remainder probleminde ise FA hariç diğer algoritmaların başarılı sonuçlar ürettiği görülmektedir. ABC algoritması leap probleminde, PSO algoritması quadratic probleminde ve DE algoritması da triangle problemlerinde başarılı sonuçlara erişmiştir. Optimum değerlerin dar bir aralığa sıkıştığı mark probleminde ise ABC ve DE algoritmaları benzer başarılı sonuçlar üretmiştir.

3.2.2.3. Approximation Level+Branch Distance Tabanlı Amaç Fonksiyonu Kıyaslaması

Approximation level + branch distance tabanlı amaç fonksiyonu kullanılması durumunda elde edilen sonuçlar Tablo 3.8'da verilmiştir. Tablo 3.8'daki sonuçlara göre [-100,100] aralığında ABC algoritması en iyi sonuçları üreten algoritma olmuştur. Arama uzayı [-1000,1000] olduğunda even-odd ve largest problemlerinde bütün algoritmalar benzer sonuçlar üretmiştir. Remainder probleminde FA algoritması diğer algoritmalara göre yetersiz kalmıştır. ABC algoritması leap ve mark problemlerinde başarılı olurken PSO algoritması ise triangle ve quadratic algoritmalarında başarılı sonuçlar elde etmiştir. Çalışma zamanına bakıldığında ise FA algoritması optimuma eriştiği zaman hızlı sonuç

üreten algoritma olmuştur. Optimuma erişemediğinde ise çevrime devam ettiği için yavaş kalmıştır.

3.2.3. Amaç Fonksiyonu Analizi

Kontrol parametrelerinin belirlenmesi gibi yeterli sonuçları elde etmek için amaç fonksiyonunun da doğru seçilmesi gerekmektedir. İyi bir amaç fonksiyonu algoritmaların hızlı ve doğru bir şekilde çözüme ulaşmasını sağlamaktadır. Amaç fonksiyonlarının rankları Tablo 3.9'da verilmiştir. Arama uzayı [-100,100] iken approximation level + branch distance tabanlı amaç fonksiyonu bütün algoritmalarda en uygun sonucu üreten amaç fonksiyonu olmuştur. Arama uzayı [-1000,1000] olduğunda ise benzerlik tabanlı amaç fonksiyonu ABC algoritması için en efektif sonuçlar üretmesine yardımcı olurken yol tabanlı amaç fonksiyonu da PSO algoritması için efektif sonuçlar üretmesine yardımcı olmuştur. Bütün algoritmaları dikkate aldığımızda ise approximation level + branch distance tabanlı amaç fonksiyonu ile genel olarak oldukça iyi sonuçlar üretilebilmektedir.

3.2.4. Rastgele Arama ile Kıyaslama

Rastgele arama (RS), rastgele test değerleri üretildikten sonra kapsama miktarlarının elde edildiği test yöntemidir. RS yönteminde kapsama miktarı maksimuma eriştiğinde veya maksimum döngü sayısına erişildiğinde (meta-sezgisel algoritmalardaki değerlendirme sayısı kadar, $250x20$) çalışma durdurulmuştur.

Tablo 3.10'da verilen RS'ye ait sonuçlar, Tablo 3.8'da approximation level + branch distance amaç fonksiyonu için verilen meta-sezgisel algoritmaların sonuçları ile kıyaslanmıştır. Tablolara göre, [-100,100] aralığında even-odd ve largest problemlerinde RS meta-sezgisel algoritmalar ile aynı sonuçları üretmiştir. Quadratic, remainder ve mark problemlerinde FA algoritmasından daha iyi sonuçlar üretmekle birlikte ABC, DE ve PSO algoritmalarından daha kötü sonuçlar üretmiştir. Leap probleminde ise ABC, RS'den daha iyi sonuçlar elde ederken, diğer algoritmalar ise RS'den daha başarısız olmuştur. Arama uzayı [-1000,1000] olduğunda ise even-odd, largest ve remainder problemlerinde ABC, DE, PSO ve RS benzer sonuçlar üretmiştir. Triangle problemi ve leap probleminde RS en başarısız algoritma olmuştur. Mark probleminde ABC algoritması en iyi algoritma

Tablo 3.9. Temel problemler için amaç fonksiyonlarının rank değerleri

	Program	AL+BD		Yol Tabanlı		Benzerlik Tabanlı	
		[-100,100]	[-1000,1000]	[-100,100]	[-1000,1000]	[-100,100]	[-1000,1000]
ABC	Triangle	1	1	2	3	3	2
	Even Odd	1	1	1	1	1	1
	Largest	1	1	1	1	1	1
	Leap Year	1	1	1	1	1	1
	Quadratic Equation	2	3	3	2	1	1
	Remainder	1	1	1	1	1	1
	Mark	1	2	2	1	1	1
TOPLAM		8	10	11	10	9	8
		18		21		17	
PSO	Triangle	1	1	3	2	2	3
	Even Odd	1	1	1	1	1	1
	Largest	1	1	1	1	1	1
	Leap Year	2	2	1	1	2	2
	Quadratic Equation	2	1	1	1	1	2
	Remainder	1	1	1	1	1	1
	Mark	1	3	3	2	2	1
TOPLAM		9	10	11	9	10	11
		19		20		21	
DE	Triangle	1	1	3	3	2	2
	Even Odd	1	1	1	1	1	1
	Largest	1	1	1	1	1	1
	Leap Year	3	2	2	1	1	3
	Quadratic Equation	1	1	2	3	2	2
	Remainder	1	1	1	1	1	1
	Mark	1	3	1	2	1	1
TOPLAM		9	10	11	12	9	11
		19		23		20	
FA	Triangle	1	1	3	2	2	3
	Even Odd	1	1	1	1	1	1
	Largest	1	1	3	2	2	2
	Leap Year	2	1	1	3	3	2
	Quadratic Equation	1	2	2	2	2	1
	Remainder	1	1	1	2	1	2
	Mark	1	3	3	2	2	1
TOPLAM		8	10	14	14	13	12
		18		28		25	
GENEL TOPLAM		34	40	47	45	41	42
		74		92		83	

olurken ikinci sırada ise RS gelmektedir. Quadratic probleminde PSO en iyi algoritma iken RS ikinci sıradadır. Genel olarak RS basit kısıtlarda oldukça kolay, hızlı ve efektif sonuçlar üretmektedir. Fakat kısıtlar zorlaştığı zaman yetersizleşmektedir.

3.3. Döngü İçeren Problemler

Bu bölümde döngü içeren problemler olarak kullanılan kod parçacıklarının karakteristik özellikleri verilmiştir.

- **Asal Sayılar:** Bu program aldığı bir değerin asal sayı olup olmadığını kontrol etmektedir.

Tablo 3.10. Temel problemler için RS algoritmasının sonuçları.

Program	[-100,100]		[-1000,1000]	
	Kapsama	Zaman	Kapsama	Zaman
Triangle	94.0404 ∓ 1.2541 (93.9394)	2.3245 ∓ 0.6504 (2.2034)	88.1818 ∓ 4.4558 (84.8485)	3.3958 ∓ 0.5430 (3.6459)
Even Odd	100.0000 ∓ 0.0000 (100.0000)	0.0135 ∓ 0.0010 (0.0135)	100.0000 ∓ 0.0000 (100.0000)	0.0115 ∓ 0.0059 (0.0098)
Largest	100.0000 ∓ 0.0000 (100.0000)	0.0327 ∓ 0.0591 (0.0215)	100.0000 ∓ 0.0000 (100.0000)	0.0249 ∓ 0.0041 (0.0237)
Leap Year	100.0000 ∓ 0.0000 (100.0000)	0.0787 ∓ 0.0816 (0.0547)	70.6667 ∓ 8.7175 (76.7273)	3.1564 ∓ 0.2946 (3.2430)
Quadratic Equation	94.6667 ∓ 5.0742 (90.0000)	1.3206 ∓ 0.5124 (1.5932)	89.3333 ∓ 2.5371 (90.0000)	2.4085 ∓ 0.5194 (2.1617)
Remainder	100.0000 ∓ 0.0000 (100.0000)	0.0300 ∓ 0.0437 (0.0197)	100.0000 ∓ 0.0000 (100.0000)	0.0450 ∓ 0.0331 (0.0315)
Mark	100.0000 ∓ 0.0000 (100.0000)	0.4760 ∓ 0.3765 (0.3104)	60.8000 ∓ 16.3209 (68.0000)	8.6096 ∓ 0.9578 (9.0612)

Tablo 3.11. Döngü içeren problemlerin karakteristikleri.

Problem	Satır Sayısı	Döngüsel Karmaşa	CFG Düğüm Sayısı	Değer Aralığı
Asal Sayılar	6	3	8	[-100,100]
Arms. Sayıları	14	3	12	[0,350]
Pal. Sayılar	13	3	11	[0,30000]

- **Armstrong Sayıları:** Bu program aldığı bir değerin armstrong sayısı olup olmadığını kontrol etmektedir.
- **Palindrom Sayılar:** Bu program aldığı bir değerin palindrom sayı olup olmadığını kontrol etmektedir.

Döngü içeren problemler olan asal sayılar, armstrong sayıları ve palindrom sayıların karakteristik özellikleri Tablo 3.11'de verilmiştir.

3.3.1. Parametre Analizi

Temel problemlerde belirtildiği üzere seçilen kontrol parametreleri algoritmaların performanslarını oldukça etkilemektedir. Burada da bütün algoritmalar için popülasyon büyüklüğü 20 ve her bir yol için de 250 döngü sayısı sabit olarak seçilmiştir. ABC algoritmasının beş farklı limit değerinde algoritmanın performansı gözlenmiştir. PSO algoritması için inertia weight (w), social compenent (c_1) ve cognitive compenent (c_2)

kontrol parametreleri için 252 farklı durum denenmiştir. DE algoritması crossover rate (CR) ve scaling factor (F) kontrol parametreleri için 48 farklı kontrol parametresi test edilmiştir. FA algoritmasının α, β ve γ kontrol parametreleri için test edilen kontrol parametresi sayısı ise 2000 tanedir. Test edilen kontrol parametrelerinin değerleri Tablo 3.12'da görülmektedir.

Tablo 3.12. Test edilen kontrol parametreleri, NP yiyecek kaynağı sayısı ve D problemin boyutunu belirtmektedir.

ABC	PSO	DE	FA
$limit$=$\{ NP*D*0.2$, $NP*D*0.4$, $NP*D*0.6$, $NP*D*0.8$, $NP*D \}$	$w = \{0.4, 0.5, 0.6, ..., 1\}$ $c_1 = \{1, 1.2, 1.4, ..., 2\}$ $c_2 = \{1, 1.2, 1.4, ..., 2\}$	$F = \{0.5, 0.6, 0.7, ..., 1\}$ $CR = \{0.3, 0.4, 0.5, ..., 1\}$	$\alpha = \{0.1, 0.2, 0.3, ..., 1\}$ $\beta_{min} = \{0.1, 0.2, 0.3, ..., 1\}$ $\gamma = \{0.5, 1, 1.5, ..., 10\}$

Her bir algoritma döngü içeren kod parçaları üzerinde farklı kontrol parametreleri ile 30 defa çalıştırılmıştır. Kapsama sonuçlarına göre önerilen parametre değerleri Tablo 3.13'te verilmiştir. Buna göre ABC algoritması için $NP * D$; PSO algoritması için $w : 0.5$, $c_1 : 1.4$, $c_2 : 1.6$; DE algoritması için $CR : 0.9$, $F : 0.8$ ve son olarak FA algoritması için $\alpha : 0.6$, $\beta_{min} : 0.9$, $\gamma : 7.5$ değerleri önerilmiştir. Bütün sonuçların ortalama, standart sapma ve medyan değerleri ise Tablo 3.14'te verilmiştir. İlk satırda ortalama ve standart sapma değerleri, ikinci satırda ise medyan değeri verilmektedir.

Tablo 3.13. Döngü içeren problemlerde kullanılan her bir algoritma için önerilen kontrol parametreleri.

ABC	PSO	DE	FA
$Limit$: $NP * D$	w: 0.5 c_1: 1.4 c_2: 1.6	CR: 0.9 F: 0.8	α: 0.6 β_{min}: 0.9 γ: 7.5

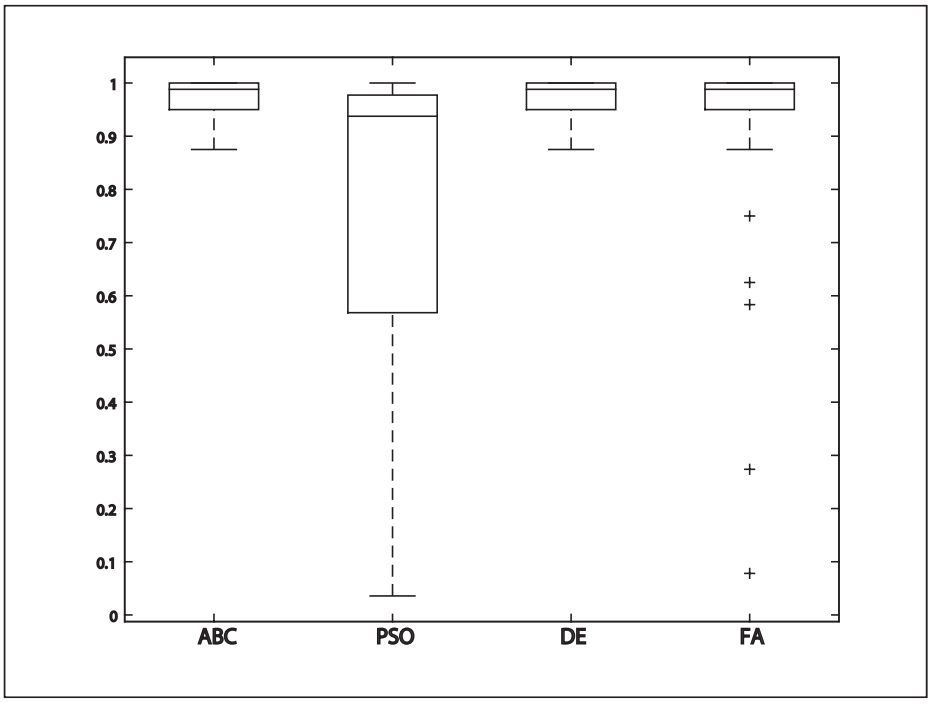
Tablo 3.14. Döngü içeren problemler için farklı ayarlarda koşturulan algoritmaların istatistiksel sonuçları. Ortalama $\mp$ Standart Sapma and Medyan değerleri verilmiştir.

Program	ABC	PSO	DE	FA
Asal Sayılar	97.8644 ∓ 0.1799 (97.9004)	75.5263 ∓ 26.5407 (86.0703)	97.7380 ∓ 0.4302 (97.9004)	97.7969 ∓ 0.2032 (97.8778)
Armstrong Sayıları	99.9333 ∓ 0.2843 (100.0000)	84.2963 ∓ 2.4567 (83.4392)	87.6042 ∓ 5.1469 (85.8333)	99.3811 ∓ 0.7309 (99.5671)
Palindrom Sayılar	83.6325 ∓ 0.0456 (83.6409)	80.2569 ∓ 2.8523 (79.1870)	83.3218 ∓ 0.6802 (83.6409)	83.6373 ∓ 0.0192 (83.6409)

Tablo 3.14 incelendiğinde asal sayılar problemi için medyan değerlerine göre en iyi sonucu ABC ve DE algoritmalarının ürettiği görülmektedir. FA algoritması da küçük bir

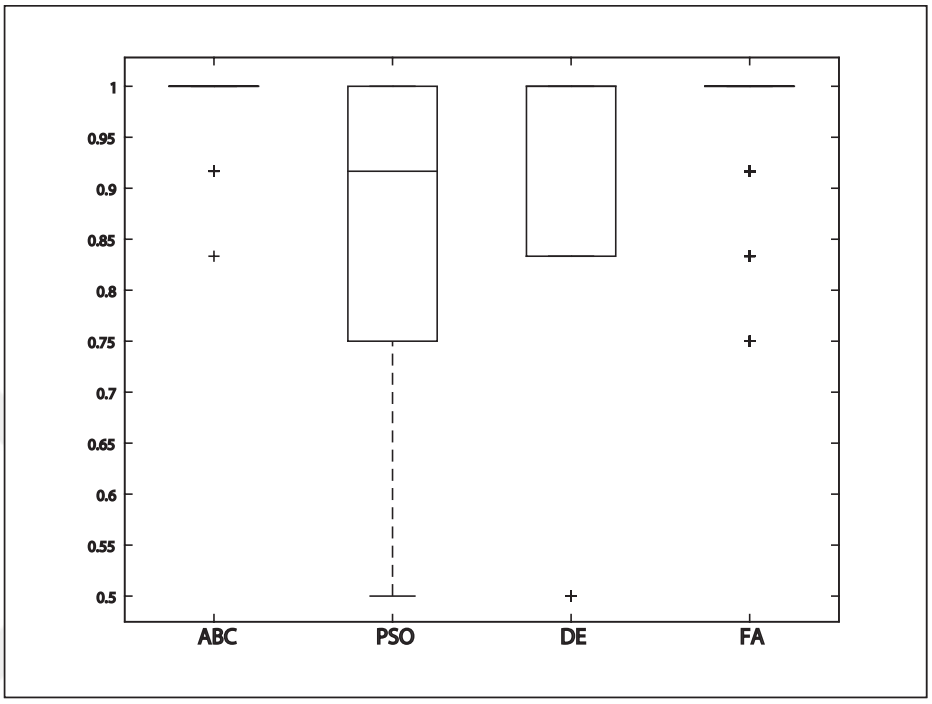
fark ile ikinci sırada gelmektedir. PSO algoritması ise efektif sonuç üretme konusunda yetersiz kalmıştır. ABC ve DE algoritmaları incelendiğinde ise ABC algoritması ortalama ve standart sapma değerlerine göre daha iyi sonuçlar elde etmiştir. Aynı şekilde DE algoritması ortalama değer ve standart sapmaya göre FA algoritmasından da biraz daha kötü sonuçlar elde etmiştir. Armstrong sayılar incelendiğinde ise medyan, ortalama ve standart sapma değerlerine göre ABC algoritmasının üstünlüğü görülmektedir. İkinci sırada ise FA algoritması gelmiştir. DE ve PSO algoritması bu problemin çözümünde yetersiz olsa da DE algoritması PSO algoritmasından biraz daha iyi performans göstermiştir. Palindrom sayılar problemi incelendiğinde medyan değerlerine göre ABC, DE ve FA algoritmaları PSO algoritmasından daha iyi performans göstermiştir. Ortalama ve standart sapma değerlerine göre ise FA algoritması ABC algoritmasından bir adım öndedir. Aynı şekilde ABC algoritması da DE algoritmasından daha iyi sonuçlar üretmiştir. PSO algoritması bu problemde de yetersiz kalmıştır.

Şekil 3.4 incelendiğinde asal sayılar problemi için PSO algoritmasının standart sapması büyük olduğu için oldukça kararsız olduğu görülmektedir. Aynı zamanda ortalama değeri de diğer algoritmalara göre geride kalmıştır. ABC, DE ve FA algoritmaları ise hemen hemen benzer davranışlar göstermiştir.



Şekil 3.4. Asal sayılar kapsama miktarı.

Şekil 3.5 incelendiğinde asal sayılar probleminde başarılı olan DE algoritmasının armstrong sayıları probleminde standart sapmasının arttığı görülmüştür. PSO algoritması ise bu problemde de iyi sonuçlar elde edememiştir. ABC ve FA algoritmaları ise benzer davranışlar göstermiştir. Şekil 3.6 incelendiğinde ise bütün algoritmaların performansı



Şekil 3.5. Armstrong sayıları kapsama miktarı.

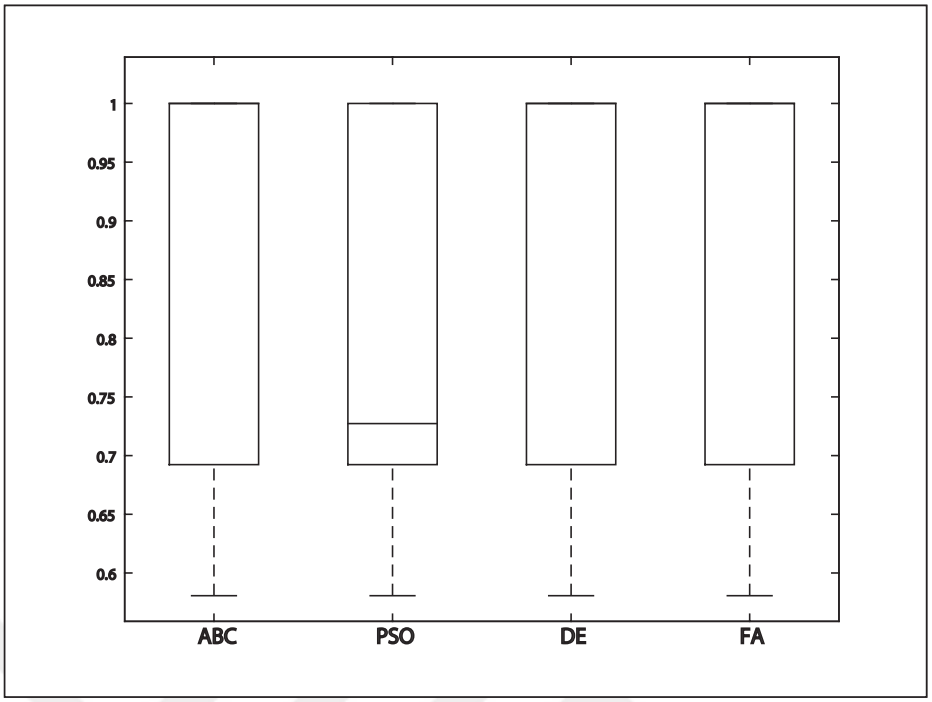
düşmüştür fakat ABC, DE ve FA algoritmalarının sonuçları PSO algoritmasına göre daha iyiye yakındır.

3.3.2. Meta-sezgisel Algoritmaların Kıyaslanması

Asal sayılar, armstrong sayıları ve palindrom sayılar problemlerinde çözüme ulaşmak için yol tabanlı, benzerlik tabanlı ve approximation level + branch distance amaç fonksiyonları ile sonuca ulaşılmıştır. Elde edilen sonuçlar yol tabanlı için Tablo 3.15 ve approximation level + branch distance için Tablo 3.16'da verilmiştir. Döngü içeren problemlerdeki yollarda tekrar eden yollar var olduğu için benzerlik tabanlı amaç fonksiyonu kullanılamamaktadır.

3.3.2.1. Yol Tabanlı Amaç Fonksiyonu Kıyaslaması

Yol tabanlı amaç fonksiyonu kullanılması durumunda elde edilen sonuçlar Tablo 3.15'te



Şekil 3.6. Palindrom sayılar kapsama miktarı.

verilmektedir.

Tablo 3.15 incelendiğinde; asal sayılar probleminde medyan değerlerine göre ABC, DE ve FA algoritmaları aynı sonuçları üretmiştir. PSO algoritması ise iyi sonuç üretmekte yetersiz kalmıştır. Ortalama değerlerine göre incelendiğinde ise FA algoritmasının az bir fark ile ABC algoritmasının önünde olduğu görülmektedir. Aynı şekilde ABC algoritması da küçük bir fark ile DE algoritmasının önünde bulunmaktadır. Çevrim sayısı ve çalışma zamanı incelendiğinde ise ABC algoritmasının DE ve FA algoritmalarından daha az çevrim sayısı ile daha hızlı sonuca eriştiği görülmektedir. FA algoritması da DE algoritmasından daha hızlı sonuca eriştiği görülmektedir. PSO algoritması ise sonuca erişememesi ve dolayısıyla çevrim sayılarını tamamlaması nedeniyle oldukça yavaş kalmıştır. Armstrong sayıları problemi incelendiğinde hem medyan hem de ortalamaya göre ABC algoritması diğer algoritmaları gerisinde bırakmıştır. Çok daha az iterasyon sayısı ve çok daha hızlı bir şekilde sonuca eriştiği görülmektedir. DE algoritması ise FA ve PSO algoritmalarını gerisinde bırakmıştır. FA ve PSO algoritmalarının ise iyi sonuçlar elde edemediği görülmektedir. Son olarak palindrom sayılar problemi incelendiğinde medyan değerlerine göre ABC ve DE algoritmaları en başarılı sonuçlara erişen algoritmalar olarak görülmektedir. Ortalama değere göre ise ABC algoritması

Tablo 3.15. Yol tabanlı amaç fonksiyonu ile döngü içeren problemler için elde edilen sonuçlar.

		Kapsama	Çevrim	Zaman
ABC	Asal Sayılar	0.9785 ∓ 0.0029 (0.9790)	128.6458 ∓ 2.6627 (128.1250)	0.6246 ∓ 0.0212 (0.6214)
	Armstrong Sayıları	$\mathbf{0.9989 \mp 0.0061}$ $(\mathbf{1.0000})$	9.0200 ∓ 8.7124 (6.0000)	0.0489 ∓ 0.0465 (0.0330)
	Palindrom Sayılar	$\mathbf{0.8364 \mp 0.0000}$ $\mathbf{(0.8364)}$	126.1933 ∓ 1.0128 (125.7500)	0.6372 ∓ 0.0103 (0.6348)
PSO	Asal Sayılar	0.8904 ∓ 0.1551 (0.9484)	241.7000 ∓ 30.9731 (250.0000)	1.1192 ∓ 0.1606 (1.1548)
	Armstrong Sayıları	0.8489 ∓ 0.0204 (0.8500)	140.4400 ∓ 29.8528 (150.4000)	0.6522 ∓ 0.1500 (0.6925)
	Palindrom Sayılar	0.8068 ∓ 0.0288 (0.7875)	159.5300 ∓ 35.8111 (175.3000)	0.7628 ∓ 0.1729 (0.8274)
DE	Asal Sayılar	0.9780 ∓ 0.0032 (0.9790)	132.2417 ∓ 16.8915 (127.0000)	0.8508 ∓ 0.0974 (0.8213)
	Armstrong Sayıları	0.9267 ∓ 0.0684 (0.9667)	24.1600 ∓ 40.8477 (8.2000)	0.1265 ∓ 0.2084 (0.0500)
	Palindrom Sayılar	0.8322 ∓ 0.0106 (0.8364)	127.8033 ∓ 6.0239 (125.5000)	0.6746 ∓ 0.0389 (0.6613)
FA	Asal Sayılar	$\mathbf{0.9790 \mp 0.0000}$ $\mathbf{(0.9790)}$	131.9125 ∓ 5.7686 (130.3125)	0.7058 ∓ 0.0325 (0.6978)
	Armstrong Sayıları	0.8672 ∓ 0.0481 (0.8833)	116.5067 ∓ 48.5548 (104.6000)	0.4821 ∓ 0.2017 (0.4357)
	Palindrom Sayılar	0.8066 ∓ 0.0231 (0.8000)	150.4667 ∓ 26.8956 (150.4500)	0.6467 ∓ 0.1272 (0.6391)

DE algoritmasının önünde bulunmaktadır. FA algoritması medyan sonuçlarına göre az farkla PSO algoritmasından daha iyi sonuç ürettiği görülmektedir. Bu problemde de PSO algoritması en başarısız algoritma olmuştur.

3.3.2.2. Approximation Level + Branch Distance Tabanlı Amaç Fonksiyonu Kıyaslaması

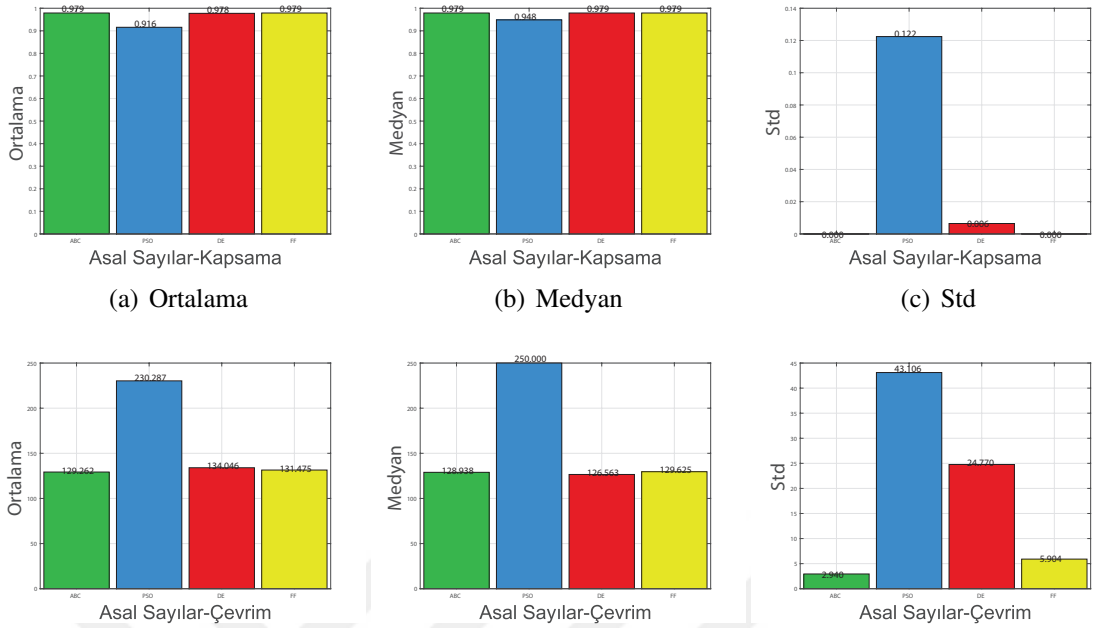
Approximation level + branch distance metriğinin amaç fonksiyonu olarak kullanılması durumunda elde edilen sonuçlar tablo 3.16'da verilmektedir. Şekil 3.7, 3.8 ve 3.9'da Tablo 3.16 özetlenmektedir. Tablo 3.16'deki asal sayılar ve Şekil 3.7 incelendiğinde; asal sayılar probleminin medyan değerine göre ABC, DE ve FA algoritmaları aynı sonuçları üretmiştir. Ortalama değerlere baktığımızda ise ABC ve FA algoritmalarının az fark ile DE algoritmasından başarılı olduğu görülmektedir. ABC algoritması da

Tablo 3.16. Approximation level+branch distance tabanlı amaç fonksiyonu döngü içeren problemler için elde edilen sonuçlar.

		Kapsama	Çevrim	Zaman
ABC	Asal Sayılar	**0.9790 ∓ 0.0000 (0.9790)**	129.2625 ∓ 2.9399 (128.9375)	0.9638 ∓ 0.1276 (0.9419)
	Armstrong Sayıları	**1.0000 ∓ 0.0000 (1.0000)**	4.3800 ∓ 2.8611 (3.8000)	0.0323 ∓ 0.0179 (0.0285)
	Palindrom Sayılar	**0.8364 ∓ 0.0000 (0.8364)**	125.7500 ∓ 0.3624 (125.6000)	0.7893 ∓ 0.0122 (0.7884)
PSO	Asal Sayılar	0.9157 ∓ 0.1224 (0.9484)	230.2875 ∓ 43.1061 (250.0000)	1.5397 ∓ 0.3026 (1.6652)
	Armstrong Sayıları	0.8450 ∓ 0.0164 (0.8500)	145.4200 ∓ 21.7269 (150.4000)	0.8081 ∓ 0.1256 (0.8366)
	Palindrom Sayılar	0.8099 ∓ 0.0284 (0.8046)	154.5500 ∓ 34.7077 (150.4000)	0.8985 ∓ 0.1979 (0.8690)
DE	Asal Sayılar	0.9775 ∓ 0.0065 (0.9790)	134.0458 ∓ 24.7699 (126.5625)	1.2067 ∓ 0.2281 (1.1407)
	Armstrong Sayıları	0.8883 ∓ 0.0605 (0.8667)	13.8867 ∓ 21.0417 (2.8000)	0.0942 ∓ 0.1395 (0.0209)
	Palindrom Sayılar	0.8352 ∓ 0.0038 (0.8364)	125.5200 ∓ 0.0484 (125.5000)	0.7893 ∓ 0.0036 (0.7894)
FA	Asal Sayılar	**0.9790 ∓ 0.0000 (0.9790)**	131.4750 ∓ 5.9040 (129.6250)	0.9704 ∓ 0.0542 (0.9583)
	Armstrong Sayıları	0.9956 ∓ 0.0075 (1.0000)	20.7200 ∓ 24.9728 (8.7000)	0.1196 ∓ 0.1462 (0.0490)
	Palindrom Sayılar	**0.8364 ∓ 0.0000 (0.8364)**	125.8267 ∓ 0.6342 (125.5000)	0.6695 ∓ 0.0066 (0.6675)

FA algoritmasından daha az çevrim sayısında sonuca ulaşmıştır. PSO algoritması ise dört algoritma içerisinde yetersiz kalmıştır. Armstrong sayıları problemi incelendiğinde (Şekil 3.8); medyan değerlerine göre ABC ve FA algoritmaları aynı performansı göstermiştir fakat ortalama değerde ABC algoritmasının FA algoritmasından daha iyi olduğu görülmüştür. ABC algoritmasının ardından FA algoritması gelmektedir. Üçüncü sırada DE algoritması gelmektedir. Bu problemde de PSO algoritması yetersiz kalmıştır.

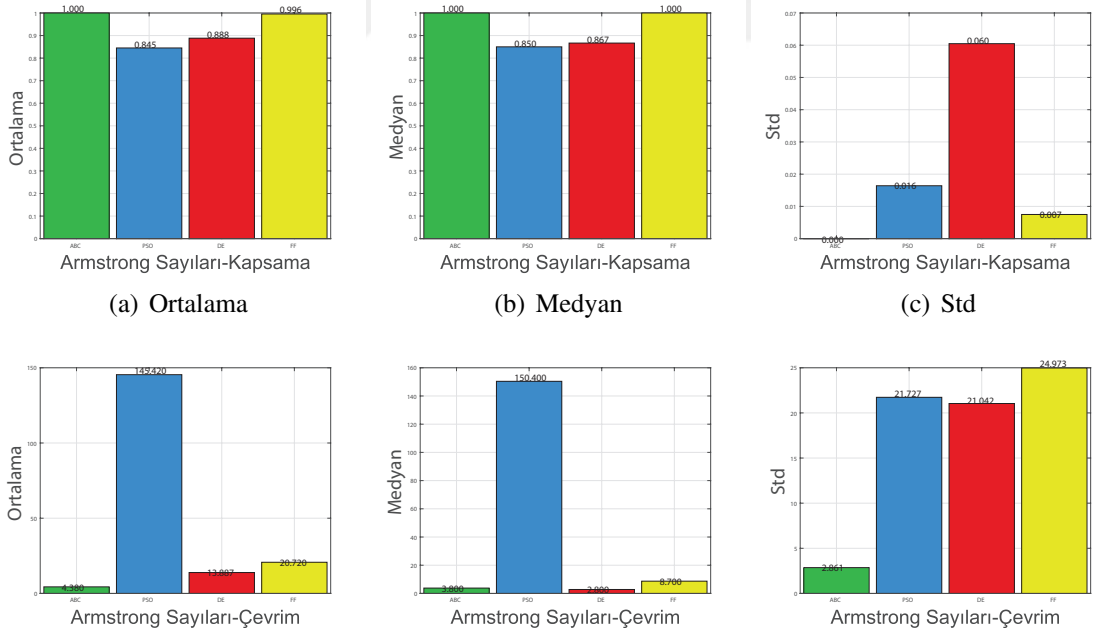
Palindrom sayılar problemi incelendiğinde ise (Şekil 3.9) medyan değerlere göre ABC, DE ve FA algoritmaları aynı sonuçları üretmiştir. Ortalama değerlere göre ise ABC ve FA algoritmalarının en iyi sonuçları elde ettiği görülmektedir. Çevrim sayısı incelendiğinde ise medyan değere göre FA, ortalama değere göre ise ABC algoritmasının üstünlüğü görülmektedir. Bu problemde FA algoritması ABC algoritmasından daha hızlı sonuca erişmiştir. Bu iki algoritmanın ardından ise DE algoritması gelmektedir. Bu problemde



(a) Ortalama (b) Medyan (c) Std

(d) Ortalama (e) Medyan (f) Std

Şekil 3.7. Asal sayılar problemi approximation level + branch distance amaç fonksiyonu ile elde edilen sonuçlar.

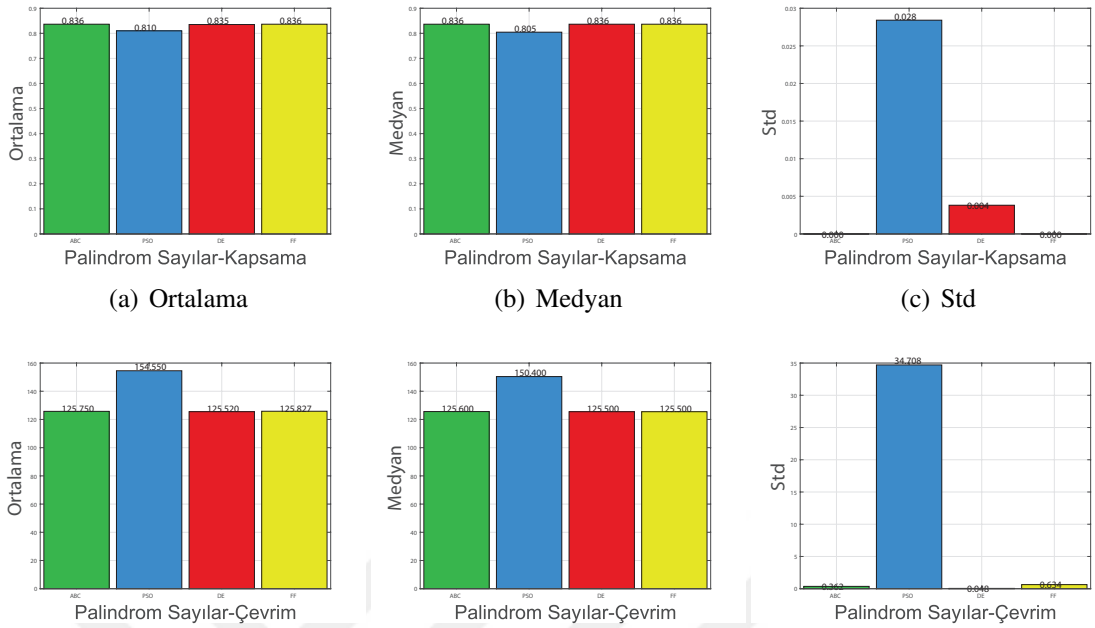


(a) Ortalama (b) Medyan (c) Std

(d) Ortalama (e) Medyan (f) Std

Şekil 3.8. Armstrong sayıları problemi approximation level + branch distance amaç fonksiyonu ile elde edilen sonuçlar.

de son sırada PSO algoritması bulunmaktadır.



(a) Ortalama (b) Medyan (c) Std

(d) Ortalama (e) Medyan (f) Std

Şekil 3.9. Palindrom sayılar problemi approximation level + branch distance amaç fonksiyonu ile elde edilen sonuçlar.

3.3.3. Amaç Fonksiyonu Analizi

Tüm amaç fonksiyonlarına ait sonuçlara göre rank değerleri Tablo 3.17'de verilmektedir. Tablo 3.17 incelendiğinde; ABC algoritması approximation level + branch distance metriklerini içeren amaç fonksiyonu ile daha iyi sonuçlar elde etmiştir. PSO algoritması asal sayılar ve palindrom sayılar probleminde approximation level + branch distance amaç fonksiyonu, armstrong sayılar probleminde ise yol tabanlı amaç fonksiyonu ile en iyi sonuçlara erişmiştir. DE algoritması asal sayılar ve armstrong sayılar probleminde yol tabanlı amaç fonksiyonu, palindrom sayılar probleminde ise approximation level + branch distance amaç fonksiyonu ile iyi sonuçlar elde etmiştir. FA algoritması ise bütün problemlerde approximation level + branch distance amaç fonksiyonu ile iyi sonuçlar elde ettiği görülmektedir. Algoritmalar genel olarak incelendiğinde ise ABC, PSO ve FA algoritmaları approximation level + branch distance amaç fonksiyonu ile DE algoritması ise yol tabanlı amaç fonksiyonu ile daha iyi sonuçlar elde etmiştir. Genel olarak ise approximation level + branch distance amaç fonksiyonunun yol tabanlı amaç fonksiyonundan daha iyi olduğu sonucu çıkarılabilmektedir.

Tablo 3.17. Döngü içeren problemler için amaç fonksiyonlarının rank değerleri.

	Program Adı	BD+AL	Yol Tabanlı
ABC	Asal Sayılar	1	2
	Armstrong Sayıları	1	2
	Palindrom Sayılar	1	2
	Toplam	3	6
PSO	Asal Sayılar	1	2
	Armstrong Sayıları	2	1
	Palindrom Sayılar	1	2
	Toplam	4	5
DE	Asal Sayılar	2	1
	Armstrong Sayıları	2	1
	Palindrom Sayılar	1	2
	Toplam	5	4
FA	Asal Sayılar	1	1
	Armstrong Sayıları	1	2
	Palindrom Sayılar	1	2
	Toplam	3	5
Genel Toplam		15	20

4. BÖLÜM

TARTIŞMA, SONUÇ ve ÖNERİLER

4.1. Tartışma, Sonuç ve Öneriler

Bu tez çalışmasında, yaygın olarak literatürde kullanılan meta-sezgisel algoritmalar yazılım mühendisliği problemlerinden olan test verisi üretimi probleminde kullanılmıştır. Alanlarındaki popülerliği ve başarılıları nedeniyle diferansiyel gelişim, parçacık sürü optimizasyonu, yapay arı koloni ve ateş böceği algoritmaları seçilmiştir. Meta-sezgisel algoritmalara ek olarak rastgele arama tekniği de çalışmada kullanılmıştır. Temel ve döngü içeren problemler olmak üzere iki farklı özelliğe sahip kod parçaları üzerinde çalışma yürütülmüştür.

Temel problemler üzerinde; meta-sezgisel algoritmalarda kontrol parametrelerinin belirlenmesinin oldukça önemli olması nedeniyle farklı kontrol parametrelerinin etkileri incelenmiştir. Çalışmada elde edilen sonuçlara göre en az kontrol parametresine sahip ABC algoritmasının en az standart sapmaya sahip olduğu gözlenmiştir. Bu da ABC algoritmasının kontrol parametresi değerlerine bağlı olmadan iyi sonuçlar üretebildiğini göstermektedir. FA algoritması ise kontrol parametrelerine en bağımlı algoritma olarak gözlenmiştir. PSO ve DE algoritmalarının da iyi kontrol parametreleri seçildiği zaman oldukça iyi sonuçlar ürettiği gözlenmiştir. Kontrol parametrelerinin analizinden sonra genel olarak iyi sonuçlar üreten bazı kontrol parametreleri önerilmiştir.

Temel problemlerin ikinci kısmında ise meta-sezgisel algoritmalar kendi içinde kıyaslanmıştır. Approximation level + branch distance, yol tabanlı amaç fonksiyonu ve benzerlik tabanlı amaç fonksiyonu olmak üzere üç farklı amaç fonksiyonundan elde edilen sonuçlar raporlanmıştır. Algoritmaların çalışması esnasında test edilen

parametreler [-100,100] ve [-1000,1000] aralığında incelenmiştir. Rastgeleliğin oldukça önemli olduğu eşitliğe dayalı dallara ayrılan problemlerde (quadratic problemi) rastgeleliği daha fazla olan algoritmalar (PSO, RS ve düşük limit değerine sahip ABC) başarılı sonuçlar üretmiştir. Kolay problemlerde (even-odd, largest, remainder) bütün algoritmalar benzer sonuçlar elde etmiştir. Leap problemi gibi birden fazla yerde optimumu bulunan multimodal problemler ve çok fazla kısıtı bulunan mark gibi problemlerde ise ABC algoritması oldukça iyi sonuçlar elde etmiştir. Uzay aralığı genişlediğinde hedef dala girme ihtimalinin düştüğü mark gibi problemlerde global arama ve yerel arama dengeli olmalıdır. ABC algoritmasındaki görevli ve gözcü arılar nedeniyle arama uzayı büyüse de algoritmanın başarısı düşmemektedir. Komşu bilgisini daha fazla kullanması nedeniyle (özellikle yüksek çaprazlama oranı ile) DE algoritması triangle probleminde başarılı sonuçlar üretmektedir. RS ise çevre bireylerdeki bilgileri kullanmadığı için triangle probleminde başarısız kalmıştır. DE ve FA gibi bölgesel yakınsamaları daha güçlü olan algoritmalar triangle gibi problemlerde daha başarılı olurken, ABC ve PSO gibi küresel araştırma yetenekleri iyi olan algoritmalar mark, quadratic equation gibi problemlerde daha başarılı olmuştur.

Temel problemlerin üçüncü kısmında ise yol tabanlı, benzerlik tabanlı ve approximation level + branch distance tabanlı amaç fonksiyonları incelenmiştir. Algoritmaların sonuç üretmesini bu amaç fonksiyonları sağladığından amaç fonksiyonunun seçimi oldukça önemlidir. Yakınsama hızı yavaş olsa da parametrelerde gerçekleşen değişikliklere göre daha hassas sonuçlar üreten approximation level + branch distance tabanlı amaç fonksiyonu oldukça iyi sonuçlar üretmiştir. ABC algoritması benzerlik tabanlı ve approximation level + branch distance tabanlı amaç fonksiyonları kullanırken iyi sonuçlar üretirken diğer algoritmalar genel olarak approximation level + branch distance tabanlı amaç fonksiyonu ile iyi sonuçlar elde etmiştir.

Çalışmanın dördüncü kısmında ise RS ile elde edilen sonuçlar raporlanmıştır. Küçük arama uzayındaki kolay kısıtlar için iyi sonuçlar üreten RS, büyük arama uzayı ve zor kısıtlarda başarısız kalmıştır.

Döngü içeren problemler irdelendiğinde ise; parametre analizi sonuçlarına göre ABC algoritması kontrol parametrelerine en az bağımlı algoritmadır. Hemen ardından ise FA

algoritması gelmektedir. Döngü içeren problemlerde kontrol parametrelerine en bağımlı algoritma ise PSO algoritmasıdır.

Döngü içeren problemlerin ikinci bölümünde ise meta-sezgisel algoritmaların kıyaslaması yapılmıştır. Approximation level + branch distance ve yol tabanlı tabanlı amaç fonksiyonları için ayrı ayrı sonuçlar irdelenmiştir. Elde edilen sonuçlara göre yol tabanlı amaç fonksiyonunda asal sayılar için ABC, DE ve FA algoritmaları benzer sonuçlar üretmiştir. Armstrong sayılar probleminde ise ABC algoritmasının en iyi sonuçları ürettiği gözlemlenmiştir. Palindrom sayılar problemi için ise ABC ve DE algoritmalarının iyi sonuçlar ürettiği görülmektedir. Bütün problemlerde PSO algoritması başarısız kalmıştır. Approximation level + branch distance tabanlı amaç fonksiyonu kullanıldığında ise asal sayılar probleminde ABC, DE ve FA algoritmaları benzer sonuçlar üretmiştir. Ancak daha hızlı sonuca ulaşma bakımından yani yakınsama hızı açısından ABC algoritması diğer algoritmaları gerisinde bırakmıştır. Armstrong sayılar problemi incelendiğinde ABC algoritması diğer algoritmalardan daha iyi sonuçlar vermiştir. Palindrom sayılar problemi incelendiğinde ise ABC ve FA algoritmaları benzer sonuçlar üretmiştir. PSO algoritması bütün problemlerde en kötü sonuçları üreten algoritma olmuştur.

Döngü içeren problemlerin üçüncü bölümünde ise amaç fonksiyonları kendi içinde kıyaslanmıştır. Elde edilen sonuçlara göre ABC, PSO ve FA algoritmaları approximation level + branch distance amaç fonksiyonu ile daha iyi sonuçlar elde etmiştir. PSO algoritması ise yol tabanlı amaç fonksiyonu ile daha iyi sonuçlar elde etmiştir. Genele bakıldığında ise approximation level + branch distance amaç fonksiyonunun yol tabanlı amaç fonksiyonundan daha iyi olduğu söylenebilmektedir.

Gelecekte bu tez çalışması genişletilerek nesne yönelimli programlama dillerinin analizinde algoritmaların davranışları incelenecektir. Bir başka çalışmada ise çok amaçlı yaklaşım ile bir amaç fonksiyonundan daha fazla amaç fonksiyonu ile sonuçlar edilmesi hedeflenmektedir. Aynı şekilde paralel yaklaşımlar kullanılarak test verisi üretiminde daha hızlı algoritmalar geliştirilebilir.

KAYNAKLAR

1. Rubenstein, D., 2007, Standish Group Report: There's Less Development Chaos.
2. Schach, S., 2010. Object-Oriented and Classical Software Engineering, McGraw-Hill Education.
3. Gürbüz, A., 2010. Yazılım Test Mühendisliği, Papatya Yayıncılık.
4. Kalıpsız, O., 2012. Sistem Analizi ve Tasarımı, Papatya Yayıncılık.
5. King, J.C., 1975. A New Approach to Program Testing, **SIGPLAN Not.**, **10**(6):228–233.
6. Godefroid, P., Levin, M.Y., and Molnar, D., 2008. Automated whitebox fuzz testing, *In NDSS*.
7. Cadar, C., Dunbar, D., and Engler, D., 2008. KLEE: Unassisted and Automatic Generation of High-coverage Tests for Complex Systems Programs, *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, 209–224, USENIX Association, Berkeley, CA, USA.
8. Khurshid, S., Păsăreanu, C.S., and Visser, W., 2003. Generalized Symbolic Execution for Model Checking and Testing, *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'03, 553–568, Springer-Verlag, Berlin, Heidelberg.
9. Chen, T., Leung, H., and Mak, I., 2005. Adaptive Random Testing, M. Maher, Ed:, Advances in Computer Science - ASIAN 2004. Higher-Level Decision Making, volume 3321 of Lecture Notes in Computer Science, 320–329, Springer Berlin Heidelberg.
10. Chen, T.Y., Kuo, F.C., Merkel, R.G., and Tse, T.H., 2010. Adaptive Random Testing: The ART of Test Case Diversity, **J. Syst. Softw.**, **83**(1):60–66.
11. Harman, M. and Jones, B.F., 2001. Search-based software engineering, **Information and Software Technology**, **43**(14):833 – 839.

12. Miller, W. and Spooner, D., 1976. Automatic Generation of Floating-Point Test Data, **Software Engineering, IEEE Transactions on**, **SE-2**(3):223–226.

13. 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.

14. Korel, B., 1990. Automated Software Test Data Generation, **IEEE Trans. Softw. Eng.**, **16**(8):870–879.

15. 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.

16. Xanthakis, S., Ellis, C., Skourlas, C., Gal, A.L., Katsikas, S., and Karapoulios, K., 1992. Application of Genetic Algorithms to Software Testing, *International Conference on Software Engineering*.

17. Harman, M. and McMinn, P., 2010. A Theoretical and Empirical Study of Search-Based Testing: Local, Global, and Hybrid Search, **IEEE Trans. Softw. Eng.**, **36**(2):226–247.

18. McMinn, P., Harman, M., Lakhotia, K., Hassoun, Y., and Wegener, J., 2012. Input Domain Reduction through Irrelevant Variable Removal and Its Effect on Local, Global, and Hybrid Search-Based Structural Test Data Generation, **Software Engineering, IEEE Transactions on**, **38**(2):453–477.

19. Michael, C.C., McGraw, G., and Schatz, M.A., 2001. Generating Software Test Data by Evolution, **IEEE Trans. Softw. Eng.**, **27**(12):1085–1110.

20. Tonella, P., 2004. Evolutionary Testing of Classes, **SIGSOFT Softw. Eng. Notes**, **29**(4):119–128.

21. Wegener, J. and Bühler, O., 2004. Evaluation of Different Fitness Functions for the Evolutionary Testing of an Autonomous Parking System, K. Deb, Ed:, Genetic and Evolutionary Computation – GECCO 2004, volume 3103 of Lecture Notes in Computer Science, 1400–1412, Springer Berlin Heidelberg.

22. Wegener, J. and Grochtmann, M., 1998. Verifying Timing Constraints of Real-Time Systems by Means of Evolutionary Testing, **Real-Time Systems**, **15**(3):275–298.

23. Derderian, K., Hierons, R.M., Harman, M., and Guo, Q., 2006. Automated Unique Input Output Sequence Generation for Conformance Testing of FSMs, **Comput. J.**, **49**(3):331–344.

24. Myers, G.J. and Sandler, C., 2004. The Art of Software Testing, John Wiley & Sons.

25. 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).

26. 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.

27. 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.

28. Mala, D.J., Mohan, V., and Kamalapriya, M., 2010. Automated software test optimisation framework – an artificial bee colony optimisation-based approach, **IET Software**, **4**(5):334.

29. 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.

30. 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.

31. 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.

32. 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.

33. Srivatsava, P.R., Mallikarjun, B., and Yang, X.S., 2013. Optimal test sequence generation using firefly algorithm, **Swarm and Evolutionary Computation**, **8**:44 – 53.

34. 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.

35. 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**.

36. 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.

37. 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).

38. mei Zhu, X. and feng Yang, X., 2010. Software Test Data Generation Automatically Based on Improved Adaptive Particle Swarm Optimizer, *2010 International Conference on Computational and Information Sciences*, Institute of Electrical & Electronics Engineers (IEEE).

39. 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.

40. 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.

41. Sommerville, I., 2010. Software Engineering (9th Edition), Pearson.

42. , 1990. IEEE Standard Glossary of Software Engineering Terminology, **IEEE Std 610.12-1990**, 1–84.

43. Smith, M.D. and Robson, D.J., 1990. Object-oriented programming-the problems of validation, *Software Maintenance, 1990, Proceedings., Conference on*, 272–281.

44. of Electrical, T.I. and Inc., E.E., 1991. Standard Glossary of Software Engineering Terminology (ANSI).

45. Andy, G., 2001. 2000 ACM SIGPLAN Haskell Workshop (Satellite Event of PLI 2000) Debugging Haskell by Observing Intermediate Data Structures, **Electronic Notes in Theoretical Computer Science**, **41**(1):1 –.

46. Thomas Müller, D.F. and the ISTQB WG Foundation Level, 2011, Foundation Level Syllabus.

47. Bird, D.L. and Munoz, C.U., 1983. Automatic Generation of Random Self-checking Test Cases, **IBM Syst. J.**, **22**(3):229–245.

48. McMinn, P., 2004. Search-based Software Test Data Generation: A Survey: Research Articles, **Softw. Test. Verif. Reliab.**, **14**(2):105–156.

49. Räihä, O., 2010. A survey on search-based software design, **Computer Science Review**, **4**(4):203 – 249.

50. 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.

51. 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.

52. Kennedy, J. and Eberhart, R.C., 1995. Particle swarm optimization, *1995 IEEE International Conference on Neural Networks*, volume 4, 1942–1948".

53. Storn, R. and Price, K., 1997. Differential Evolution - A Simple and Efficient Heyristic for global Optimization over Continuous Spaces, **Journal of Global Optimization**, **11**(4):341–359.

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

55. Yang, X.S., 2008. Nature-Inspired Metaheuristic Algorithms, Luniver Press.

56. Karaboga, D. and Akay, B., 2009. A Survey: Algorithms Simulating Bee Swarm Intelligence, **Artificial Intelligence Review**, **31**(1):68–55.

57. Karaboga, D., Gorkemli, B., Ozturk, C., and Karaboga, N., 2014. A comprehensive survey: artificial bee colony (ABC) algorithm and applications, **Artificial Intelligence Review**, **42**(1):21–57.

58. Akay, B. and Karaboga, D., 2015. A survey on the applications of artificial bee colony in signal, image, and video processing, **Signal, Image and Video Processing**, **9**(4):967–990.

59. 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).

60. Liang, X., Guo, S., Huang, M., and Jiao, X., 2014. Combinatorial Test Case Suite Generation Based on Differential Evolution Algorithm, **JSW**, **9**(6).

61. Pham, D.T. and Karaboga, D., 2000. Intelligent Optimisation Techniques, Springer Science Business Media.

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

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

64. Stephan Olariu, A.Y.Z., 2005. Handbook of Bioinspired Algorithms and Applications (Chapman & Hall/CRC Computer and Information Science Series), Chapman and Hall/CRC.

65. Baresel, A., Sthamer, H., and Schmidt, M., 2002. Fitness Function Design To Improve Evolutionary Structural Testing, *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '02, 1329–1336, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

66. Tracey, N., Clark, J., Mander, K., and McDermid, J., 1998. An Automated Framework for Structural Test-Data Generation, *Proceedings of the 13th IEEE International Conference on Automated Software Engineering*, ASE '98, 285–, IEEE Computer Society, Washington, DC, USA.

67. Lin, J.C. and Yeh, P.L., 2001. Automatic test data generation for path testing using GAs, **Information Sciences**, 47 – 64.

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı, Soyadı	:	Ömür ŞAHİN
Uyruğu	:	Türkiye (T.C.)
Doğum Tarihi ve Yeri	:	01.01.1991/Kayseri
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 Bölüm Başkanlığı 38039, Melikgazi KAYSERİ TÜRKİYE

EĞİTİM

Derece	Kurum	Mezuniyet Yılı
Lisans	ERÜ Müh. Fakültesi Bilgisayar Müh., KAYSERİ	2014

İŞ DENEYİMİ

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