ANKARA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

MUTASYON BAZLI YAZILIM TESTLERİNİN İYİLEŞTİRİLMESİ İÇİN BİR YAKLAŞIM

MOHAMED ABDILLAHI WARSAME

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

ANKARA
2019

TEZ ONAYI

Mohamed Abdillahi WARSAME tarafından hazırlanan "**Mutasyon Bazlı Yazılım Testlerinin İyileştirilmesi İçin Bir Yaklaşım**" adlı tez çalışması 27/08/2019 tarihinde aşağıdaki jüri tarafından oy birliği ile Ankara Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühesdisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Danışman : Dr. Öğr. Üyesi Özgür TANRIÖVER
Ankara Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı

Jüri Üyeleri:

Başkan : Doç. Dr. Çelebi ULUYOL
Gazi Üniversitesi Bilgisayar ve Öğretim Teknolojileri Eğitimi Anabilim Dalı

Üye : Prof. Dr. İman ASKERBEYLİ
Ankara Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı

Üye : Dr. Öğr. Üyesi Özgür TANRIÖVER
Ankara Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı

Yukarıdaki sonucu onaylarım.

Prof. Dr. Özlem YILDIRIM
Enstitü Müdür Vekili

ETİK

Ankara Üniversitesi Fen Bilimleri Enstitüsü tez yazım kurallarına uygun olarak hazırladığım bu tez içindeki bütün bilgilerin doğru ve tam olduğunu, bilgilerin üretilmesi aşamasında bilimsel etiğe uygun davrandığımı, yararlandığım bütün kaynakları atıf yaparak belirttiğimi beyan ederim.

27/08/2019

Mohamed Abdillahi WARSAME

ÖZET

Yüksek Lisans Tezi

MUTASYON BAZLI YAZILIM TESTLERİNİN İYİLEŞTİRİLMESİ İÇİN BİR YAKLAŞIM

Mohamed Abdillahi WARSAME

Ankara Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Dr. Öğr. Üyesi Ömer Özgür TANRIÖVER

Mutasyon testi, hataları ortaya çıkarmak ve tespit etmek için etkili olduğu bilinen bir tür beyaz kutu testidir. Hata tespitinde test kümesinin yeterliliğini değerlendirmek için tasarlanmış en güçlü tekniklerden biridir. Bununla birlikte, mutasyon testi, yüksek sayıda mutant oluşturulduğundan, eşdeğer mutantları tespit etme çabası ve test grubunun gerçeğinin, her mutant için gerçekleştirilmesi gerekebileceği için maliyetlidir. Selektif mutasyon testi gibi bu problemlerin üstesinden gelmek için yöntemler geliştirilse de, özellikle anlamsal olarak eşdeğer mutantların tespit edilmesi ve elimine edilmesi sorunu hala devam etmektedir. Bu tez, ilk olarak eşdeğer mutantları tespit edip ortadan kaldırmak için test vakalarına öncelik tanıyan ve daha sonra, test vakalarının yürütülmesi ile çok sayıda mutantı kademeli olarak azaltmak için, aralarındaki benzerliklere göre mutantları farklı gruplara kümeleyen bir yöntem sunar. Mutasyon kümelenmesi ve yok edilmesinde çeşitli çalışmalar yapılmış olmasına rağmen, bu çalışmalarda öncelikli ve eşdeğerliğe dayalı test durumlarının kaldırılması göz önünde bulundurulmamıştır. Deney sonuçları, yöntemin saf mutasyon testi kadar etkili bir şekilde mutasyon skoru olduğunu ve özellikle program ünitesinin boyutu ve mutasyonu arttıkça uygulama süresini kısalttığını göstermiştir.

Ağustos 2019, 61 sayfa

Anahtar Kelimeler: Mutasyon testi, test vakası önceliği, mutasyon kümelemesi, seçmeli test

ABSTRACT

Master Thesis

AN APPROACH FOR IMPROVING MUTATION BASED SOFTWARE TESTING

Mohamed Abdillahi WARSAME

Ankara University
Graduate School of Natural and Applied Sciences
Department of Computer Engineering

Supervisor: Asst. Prof. Dr. Ömer Özgür TANRIÖVER

Mutation testing is a type of white box testing which has been found to be effective at revealing and detecting faults. It's one of the most powerful techniques designed to evaluate the adequacy of test suite in detecting faults. However, mutation testing is expensive due to high number of mutants created, the effort to detect equivalent mutants and the fact of test suite may have to be executed for each mutant. Even though methods have been developed to overcome these problems such as selective mutation testing, the problem of detecting and eliminating especially semantically equivalent mutants still remains. This thesis presents a method which first gives priority to test cases according to their reach to detect and eliminate equivalent mutants and then clusters mutants into different groups according to similarities among them to reduce progressively the large number of mutant vs test case executions. Although various studies have been conducted in mutation clustering and elimination, eliminating test case executions based on priority and equivalence have not been studied previously. The experimental results showed that, the method is mutation score wise comparable to pure mutation testing and reduces execution time especially when the size and the number of mutations of the program unit increases.

August 2019, 61 pages

Keywords: Mutation testing, test case priority, mutation clustering, selective testing

TEŞEKKÜRLER

Her şeyden önce yüce Allah'a şükranlarımı sunuyorum. Bu zorlu dönemde bana her türlü desteği verip vaktini ayıran danışman hocam Dr. Öğr. Üyesi Ömer Özgür TANRIÖVER'e çok teşekkür ederim.

Mohamed Abdillahi WARSAME
Ankara, Ağustos 2019

İÇİNDEKİLER

KISALTMALAR DİZİNİ

LOC	Kod Satırı (Line of Codes)
MS	Mutasyon Skoru (Mutation Score)
TC	Test Vakası (Test Cases)

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

1. GİRİŞ

Yazılım testi, yazılımın gelişim sürecinde önemli bir faaliyettir. Yazılımın mevcut durumu ile istenen koşullar arasındaki farkları tanımlamak için kullanılan bir işlemdir. Yazılım testinin amacı, yazılımın gereksinimlerini karşılayıp karşılamadığını belirlemek için yazılımı geliştirmek ve yazılımın herhangi bir hata içermediğinden emin olmaktır. Aslında, bir yazılım proje kaynaklarının çoğu (% 50'den fazla) arızaların tespitine ve düzeltilmesine yatırım yapılmaktadır. Bu büyük yatırıma rağmen yazılımın karmaşıklığın yazılımın tüm hatalarını keşfetmeyi imkansız hale getirdiği bilinmektedir. Birçok yazılım test tekniği bilinmektedir. Bu teknikler; Kara Kutu testi, Beyaz Kutu testi ve Gri Kutu testi olarak sınıflandırılabilir. Bu tekniklerin her kategorisi farklı karmaşıklık ve kapsam seviyelerini hedefler. Beyaz kutu testi kategorisinde Mutasyon testi adı verilen güçlü bir test tekniği vardır. Bu test stratejisinin asıl işi test vakalarını tekrar tekrar iyileştirerek tüm hataları tanımlayabilen etkili test kümesi üretmektir. Mutasyon testi etkilidir ancak hesaplama açısından maliyetli olduğuna inanılmaktadır. Maliyetli olmasının nedenlerinden biri çok sayıda mutant üretmek ve bu mutantları test kümesindeki her bir test vakasına karşı çalıştırmaktır. Herhangi bir test durumuyla ulaşılamayan ve öldürülemeyen eşdeğer mutantların tespit edilmesi ve elimine edilmesi çözülmesi gereken bir başka sorundur. Bu tez, önce eşdeğer mutantları tespit eden, uygulama süresini azaltmak için bazı kriterlere göre test vakalarına öncelik veren ve daha sonra çok sayıda mutantları azaltmak için benzerlerine göre farklı gruplara kümeleyen bir yöntem sunar. Önerilen yaklaşım üçüncü bölümde kısaca açıklanmaktadır.

1.1 Mutasyon Testi

Bir tür beyaz kutu ünite testi olan mutasyon testi, hata tespitinde en etkili yöntem olarak kabul edilir. İlk kez 1971 yılında Richard Lipton [Mathur A. P 1994] tarafından tanıtılmıştır ve bir test kümesinin yeterliliğinin ölçülmesinde hataya dayalı kullanışlı bir tekniktir. Mutasyon testi “Mutasyon Yeterlilik Skoru” adı verilen bir test kriteri sağlayan hata bazlı test tekniğidir. Mutasyon yeterlilik Skoru, bir test takımının etkinliğini arıza tespit yeteneği açısından ölçmek için kullanılabilir (Yue Jia ve Mark Harmanu 2011). Mutasyon testinde, yazılımın kaynak kodunda küçük bir değişiklik yapılır. Ardından test durumlarının hataları tespit edip edemediğini kontrol etmek için

yazılım yürütülür. Hatalı programdaki değişikliklerin programın amacı üzerinde bir etkisi yoktur. Bu test tekniği orijinal bir programa ve o programın birçok hatalı versiyonuna dayanmaktadır. Bu hatalı sürümlere mutant denir. Bu mutantların her biri, programı sadece bir operasyonla mutasyona sokarak oluşturulur. Çizelge 1.1'e bakınız.

Çizelge 1.1 Bir programın bazı olası mutantları.

Orijinal Programı (*P*)	Mutant1 (*m1*)	Mutant2 (*m2*)	Mutant3 (*m3*)
a= b+c;	a= b*c;	a= b-c;	a= b+c+c;

Mutasyon testi genel düşüncesi, hataların bir programcı tarafından yapılan hataları temsil etmesidir. Bu nedenle kasıtlı olarak mutantlar adı verilen bir dizi hatalı programın oluşturulması için programa dahil edilirler. Her mutant program, orijinal programdaki bir yere bir mutant operatörü uygulanarak elde edilir. Standart mutasyon operatörler, bir operatörün değiştirilmesini içerir, mesela '+' operatörü başka bir operatör ile değiştirmek örneğin '–' veya bir değişkeni bir başkasıyla değiştirmek. Belirli bir test kümesinin kalitesini değerlendirmek için mutantlar, eklenen hataların tespit edilip edilemeyeceğini görmek için bir girdi veri kümesi üzerinde yürütülür. Test değişikliği tespit edebiliyorsa (yani testlerden birinin başarısız olması durumunda) mutantın öldürüldüğü söylenir. Test için girdi verileri mutant ve orijinal program için farklı program durumlarına neden olmalıdır (Ilona Bluemke ve Karol Kulesza. 2014). Bir dizi test vakası oluşturulması için mutant kümesi kullanılmalıdır. Amaç, tüm mutantları öldürme gücüne sahip, bir dizi test vakasına sahip olmaktır. Orjinalinden farklı bir çıktı ve en az bir mutant üretebilen test vakalarının yeterli olduğu söylenmelidir aksi halde başarısız olarak adlandırılmaktadır.

1.1.1 Mutasyon skoru

Mutasyon Skoru, öldürülen mutantların sayısının toplam mutant sayısı ile eşdeğer mutantların sayısı arasındaki farka oranıdır. MS'in değeri '0' ve '1 'arasındadır. MS daha düşük bir değere sahipse, bu mutantların test kümesi tarafından doğru bir şekilde tespit edilemeyeceği anlamına gelmektedir. MS daha yüksek bir değere sahipse, mutantların çoğunun bu test kümesi ile öldürüldüğü söylenmektedir. MS = 0 olduğunda, mutantların herhangi bir test vakası tarafından öldürülemeyeceği anlamına gelir ve MS

= 1 olduğunda mutantlar kolayca öldürülebilir demektir. MS aşağıdaki formülle hesaplanır.

$$MS = \frac{\text{Öldürülen } Mutantların\ Sayısı}{Toplam\ mutant\ sayısı - Eşdeğer\ mutant\ sayısı}$$

Bir dizi test verisinin mutasyon skoru, bu veriler tarafından öldürülen eşdeğer olmayan mutantların yüzdesidir. Bir test verisinin, mutasyon skorunun %100 olması halinde mutasyon için yeterli olduğu söylenir (Ahmed S. Ghiduk, Moheb R. Girgis, Marwa H. Shehata 2017).

1.1.2 Mutasyon işlemi

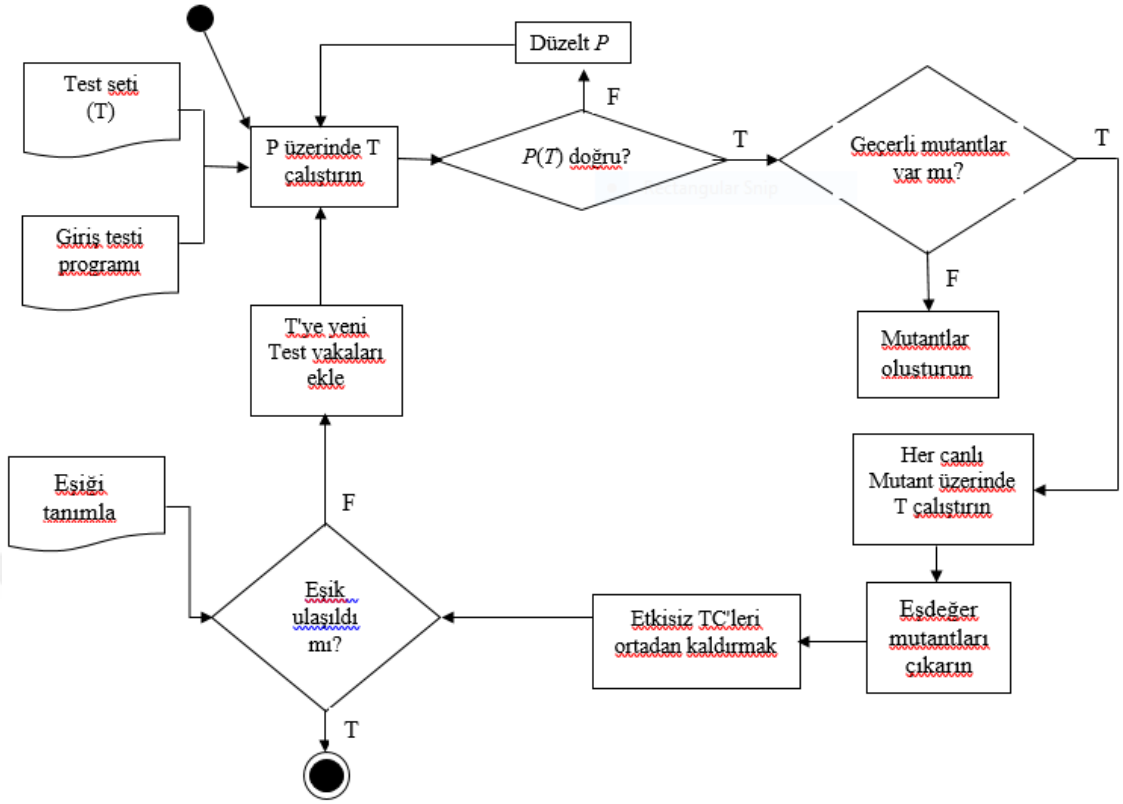
Mutasyon Testi işlemi aşağıdaki gibi üç adımda gerçekleştirilir.

- Birincisi, mutantlar adı verilen bir dizi hatalı program üretmek için basit sözdizimsel değişiklikler yaparak hataları orijinal programa yerleştirmek; her mutant, farklı bir sözdizimsel değişim içerir.

- İkincisi, orijinal programda ve ayrıca tüm mutantlarda bir dizi test vakası uygulanır. Daha sonra, bu test vakalarının hataları saptamadaki kabiliyeti mutasyon skoru (MS) hesaplanarak değerlendirilir. MS, bir testin etkinliğini arıza tespit yeteneği açısından ölçer.

- Üçüncüsü, orijinal programın ve mutasyona uğramış olanın sonuçlarını karşılaştırmak. Bir mutant uygulamasının sonucu, girdi test kümesindeki herhangi bir test vakası için orijinal programın yürütülmesinden farklıysa, mutant tarafından belirtilen hata öldürülür veya tespit edilir. Aksi takdirde hayatta olduğu söylenir.

Mutasyon test işlemi, şekil 1.1'deki akış şeması ile kolayca tanımlanabilir



Şekil 1.1 Tipik Mutasyon test sürecini temsil eden akış şeması.

Her şeyden önce, tek bir mutasyon operatörü uygulanarak orijinal program P, mutasyona uğratılır. Bir mutant operatörü uygulanarak mutantlar ve ardından test vakaları üretilir. Her test vakası *P* üzerinde çalıştırılır. Eğer P'nin çıkışı doğru değilse, bir hata bulunur ve P düzeltilir. Mutantlar üzerinde test vakası çalıştırılmadan önce P'de hata yoksa veya hatalar giderilirse test vakaları her mutant üzerinde çalıştırılır. Sonuç farklıysa arıza tespit edilir ve Çizelge 1.2'deki test vakası 1'de olduğu gibi, mutantın öldürüldüğü söylenir. Eğer testin sonucu orjinal ve mutant program için aynıysa Çizelge 1.2'deki test vakası 2'de olduğu gibi o zaman mutantın eşdeğer bir mutant (öldürülemeyen) olduğunu veya test vakasının verimli olmayıp iyileştirilmesi gerektiğini gösterir. Eşdeğer bir mutant sözdizimsel olarak farklıdır, ancak orijinal programa anlamsal olarak aynıdır. Onlar sadece hesaplama maliyetini arttırırlar ve bir test vakasının etkinliğini belirlemede yardımcı olmamakla kalmayıp herhangi bir test durumuyla asla öldürülemezler.

Çizelge 1.2 Mutantları tespit etmek için oluşturulan test vakaları

Test vakaları	*P* z=x+c	*m*1 z=x*c
Test vakası 1 x=4, c=2	z=6	z=8
Test vakası 2 x=2, c=2	z=4	z=4

1.1.3 Mutasyon operatörleri

Mutasyon operatörleri bir programın mutantlarının üretilmesinden sorumludur; Orijinal programda sözdizimsel değişiklikler yaparlar. Bu sözdizimsel değişiklikler, kodlama sırasında programcıların yaptığı olağan hataları yansıtır. Mutasyon operatörleri uyması gereken iki temel amacı vardır. Bunlar programcıların yaptığı tipik hataları tanıtmak ve her dalı uygulayarak testi zorlamak. Çizelge 1.3, Mothra'nın 22 mutasyon operatörünü göstermektedir. Mutasyon operatörleri kısaltmaları ile temsil edilir. Örneğin, "AAR" mutasyon operatörü, bir programdaki her bir dizi referansını programdaki diğer farklı bir dizi referansı ile değiştiren "dizi referans değişimi için dizi referansı" nın kısaltmasıdır (A. J. Offutt, 1996).

Çizelge 1.3 22 Mothra'nın Mutasyon operatörleri

No	Mutasyon Operatörü	Tanım
1	AAR	Array reference for array reference replacement
2	ABS	Absolute value insertion
3	ACR	Array reference for constant replacement
4	AOR	Arithmetic operator replacement
5	ASR	Array reference for scalar variable replacement
6	CAR	Constant for array reference replacement
7	CNR	Comparable for array name replacement
8	CRP	Constant replacement
9	CSR	Constant for scalar variable replacement
10	DER	DO-statement end replacement
11	DSA	DATA-statement alterations
12	GLR	GOTO label replacement
13	LCR	Logical connector replacement
14	ROR	Relational operator replacement
15	RSR	RETURN-statement replacement
16	SAN	Statement analysis
17	SAR	Scalar variable for array reference replacement
18	SCR	Scalar for constant replacement
19	SDL	Statement deletion
20	SRC	Source constant replacement
21	SVR	Scalar variable replacement
22	UOI	Unary operator insertion

1.2 Amaç

Özel, açık ve gerçekçi hedeflere sahip olmak tezin kalitesini arttırır ve hedeflerine ulaşmayı kolaylaştırır. Bu tez için belirlenen hedefler:

- Genel olarak mutasyon test tekniğini ve özellikle de bu tekniğin hesaplama açısından maliyetli olma problemini araştırmak.
- Test vakalarına öncelik vererek mutant sayısını azaltmak ve ardından mutantları K-means kümeleme algoritması ile kümelenmek.
- Yüksek mutasyon skorunu elde etmek için kümelerden seçilen mutantlardan üretilen yeterli test kümesi ve etkili bir mutasyonu almaya gayret etmek.

1.3 Problem Tanımı

Yazılımın güvenilirliğini ve kalitesini en üst düzeye çıkarmak, yazılım testinin arkasındaki ana fikirdir. Bu nedenle toplam maliyet ve zamanın yarısı yazılımı iyice test etmeye harcanmaktadır. Bunu akılda tutarak, birkaç test stratejileri getirilmiştir. Bu teknikler arasında mutasyon testlerinin, hata tespitinde en güçlü olduğu ortaya çıkmıştır. En güçlü test tekniklerinden biri olmasına rağmen, mutasyon testleri uygulanmaz ve test için yaygın olarak kullanılmaz. Hesaplamalı olarak maliyetli olmak mutasyon testinin göz ardı edilmesinin ana nedenidir.

1.3.1 Hesaplamalı maliyet

Mutasyon testi ile ilgili muazzam hesaplama maliyetinin nedenine bakıldığında mutasyon testini yaygın olarak kullanılmayan bir teknikte yapan [A. J. Offutt, R. H. 2001] 'in çalışmasında tartışıldığı gibi üretilen çok sayıda mutant ve bunları her bir test vakasına karşı çalıştırılması, hesaplamanın maliyetli olmasının temel nedenidir. Küçük bir program bile yüzlerce mutanta sahip olabilir. Örneğin 100 LOC'den daha az bir program muazzam sayıda mutant üretmektedir. 216 mutantı 48 LOC ile üreten, bu araştırmanın deneylerinde kullanılan Java Üçgen programı gibi. Test vakaları her üretilen mutanta karşı en az bir kere çalıştırmalı ve bu da çok fazla hesaplama sonucu olmaktadır. (W. E. 1993) 'de belirtildiği gibi maliyet, kriteri ölçmek için kullanılan test kümesinin büyüklüğü ve gözlemlenen mutantların sayısı olarak iki ölçümle ölçülebilir.

1.3.2 Mutasyon testinin hesaplama maliyetini azaltmanın yolları

Yukarıda bahsedildiği gibi, mutasyon testlerinin hesaplama açısından maliyetli olduğu inanılmaktadır, ancak son araştırmalar, araştırmacıların mutasyon testinin maliyetini önemli ölçüde azaltmak için çeşitli teknikler ve yöntemler bulduklarını göstermektedir. Bu yöntemler iki kategoride sınıflandırılır (Offutt ve Untch 2001). Bu kategoriler ikinci bölümde daha ayrıntılı olarak açıklanmaktadır.

- **Az yapma yaklaşımı (Do fewer approach):** Önemli bilgi kaybı olmadan mutantların sayısının azaltılması (Z. Zhao and vd. 2009, W. B. Langdon 2009).
- **Daha hızlı ve akıllı yapma yaklaşımı (Do faster and smarter approach):** mutantların hesaplarını birçok makineye dağıtarak mutantların hızlı üretimi ve çalıştırılması.

Bu tezin sınırı "daha azını yapma" yaklaşımına odaklanarak hesaplama maliyetinde düşüşe yol açan mutantların sayısını azaltmanın bir yolunu bulmaktır.

1.4 Çalışma Organizasyonu

Tezin geri kalanı aşağıdaki şekilde düzenlenmiştir. Bölüm 2 literatürdeki incelemeyi, araştırmanın mevcut durumunu, tezin başlığının anahtar sözcüklerini ve bu problem alanında halihazırda yapılan çalışmaları açıklamaktadır. Bölüm 3 tez için uyarlanmış metodoloji ile ilgilidir. Bölüm 4 deneyin değerlendirme bölümü ile ilgilidir. Bu, elde edilen sonuçları ayrıntılı olarak açıkladığı gibi, raporun önemli bir bölümüdür. Bölüm 5 tezin detaylı bir sonucunu sunar ve son olarak gelecekteki bazı ilerlemeler önererek tezi sonlandırır.

2. LİTERATÜR TARAMASI

Bu bölümde ilk olarak yazılım testi açıklanmaktadır. Ardından önerilen yaklaşımın önemli bir kısmı olan test vakası önceliklendireme hakkında bilgi verilmektedir. Daha sonra kümelenme ve özellikle de bu tezde uygulanan K-means kümelenme algoritması hakkında kısa bir açıklama verilmektedir. Son olarak problem alanında yapılan önceki çalışmaları vurgulamaktadır.

2.1 Yazılım Testi

Yazılım testinin, hata bulma amacıyla bir program / sistem yürütme işlemi olduğu söylenmiştir [G. J. Myers vd. 2004]. Ayrıca, analiz yoluyla yazılımın işlevselliğini elde etme prosedürü olarak Tanımlanmıştır [anonymous 2018d]. Yazılım testi, hata bulma programı çalıştırarak yazılım kalitesini kontrol etmek ve doğrulamak için kullanılır [Srinivas Nidhra ve Jagruthi Dondeti. 2012]. Yazılım testi, yazılım kalite güvencesinin kilit unsurlarından biridir ve şartname, tasarım ve kodlamanın bir incelemesini oluşturur. Yazılımın yakından izlendiği koşullarda sistematik olarak test edilerek yazılım sistem kalitesini doğrulamak, yazılım testinin temel amacıdır. Diğer bir amaç, yazılımın bütünlüğünü ve doğruluğunu tespit etmek ve keşfedilmemiş hataları ortaya çıkarmaktır [anonymous 2018d].

2.1.1 Yazılım test teknikleri

Yazılım testi iki ana kategoriye ayrılabilir

- Beyaz Kutu Testi
- Kara Kutu Testi

Beyaz Kutu Testi; Test yapanın, iç mantık, kodun yapısı ve nasıl kodlandığı hakkında tam olarak bilgi sahibi olduğu test tekniği beyaz kutu testi olarak bilinir. Test vakaları elde etmek için prosedürel kontrol yapısını kullanan bir test senaryosu tasarımı yöntemidir. Yazılım test sürecinin entegrasyon, birim ve sistem seviyelerinde kullanılır [Farmeena Khan ve Mohd. Ehmer Khan. 2012].

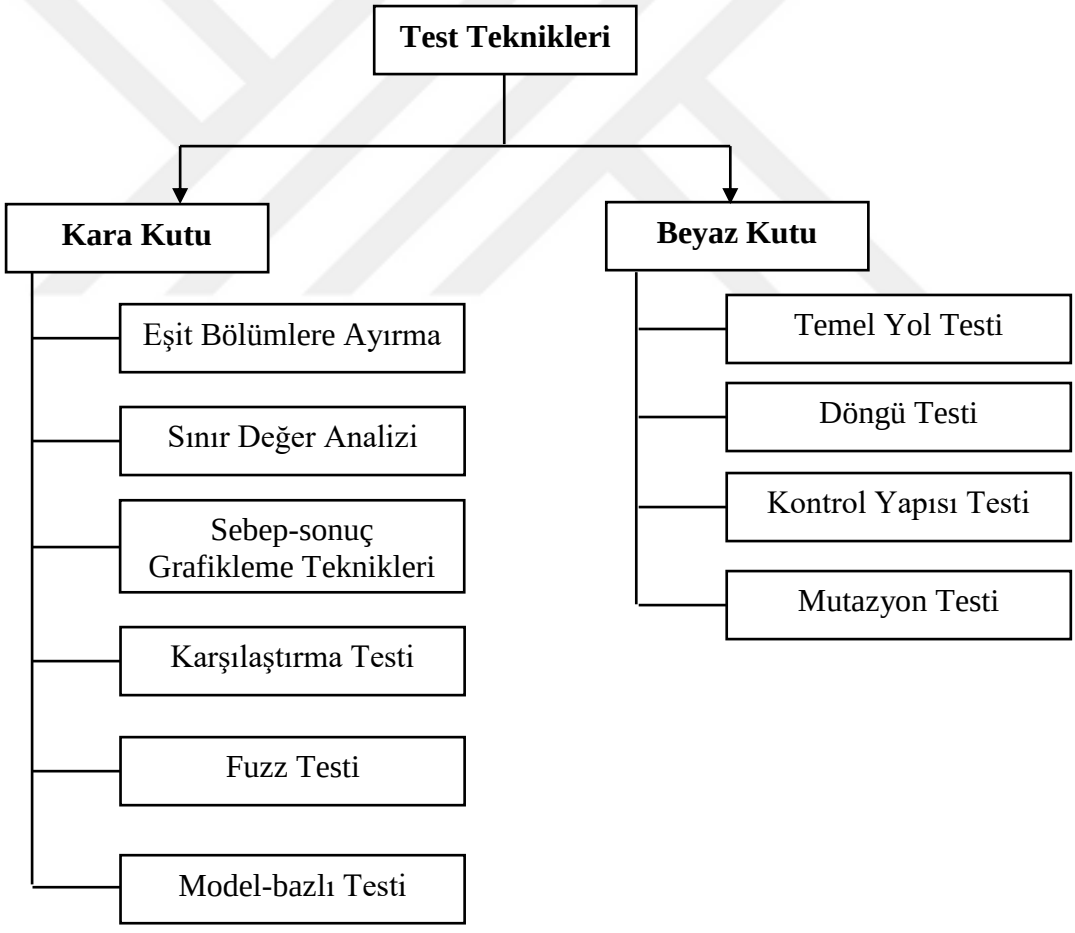
Beyaz kutu test tekniğini kullanmanın avantajları:

- Gizli koddaki hatalar, ilave kod satırları kaldırılarak gösterilir
- Bir modülde tüm bağımsız yollar en az bir kez uygulanır

- Yan etkileri olması yararlıdır
- Tüm mantıksal kararlar uygulanır
- Test senaryosu yazımı sırasında maksimum kapsam sağlanır
- Tüm döngüler sınırlarında çalıştırılır
- Geçerliliklerini korumak için iç veri yapıları kullanılır

Beyaz kutu kullanımının dezavantajları:

- Yetenekli bir test yapanın yapması gerekli olduğundan dolayı, beyaz kutu testinin maliyetli olduğuna inanılmaktadır
- Her köşe ve köşedeki gizli hataları bulma zorluğu nedeniyle, birçok denenmemiş yol kalabilmektedir
- Kodda atlanmış kodlardan bazıları eksik olabilmektedir



Şekil 2.1 Test tekniklerinin genel sınıflandırması

Kara kutu testi: Uygulamanın dahili çalışmasının bilinmediği bir yazılım test yöntemidir [Farmeena Khan ve Mohd. Ehmer Khan. 2012]. Kara kutu testinde, bir sistemin yapısı göz ardı edilir ve seçilen çıktılara cevap olarak seçilen girdiler ve yürütme koşulları odaklanır [Mohd. Ehmer Khan. 2011]. Kara kutu testinin temel amacı, yanlış veya eksik işlevleri ortaya çıkarmak, arayüz hatalarını keşfetmek, veri yapısının hatalarını veya dış veritabanlarındaki hatalarını belirlemek ve performans hatalarını bulmaktır. Birim testi, regresyon testi, entegrasyon testi, sistem testi ve kabul testi gibi yazılım testi aşamalarının seviyelerinde uygulanmaktadır.

Kara kutu testinin avantajları:

- Test vakasının geliştirilmesi hızlı ve kolaydır
- Hata sınıflarının yokluğu test durumları ile gösterilebilir
- Test yapanın anlaması çok kolaydır
- Kullanıcıların görüşleri ve geliştiricilerin bakış açısı açıkça ayrılmıştır
- Büyük kod bölümü için etkilidir.

Kara kutu testinin dezavantajları:

- Kapsam, yalnızca seçilen test senaryosu sayısının bir sonucu olarak sınırlıdır
- Açık bir spesifikasyon olmadan test senaryoları tasarlamak zordur
- Verimsiz bir testidir

Çizelge 2.1 Test spektrumu

Test tipi	**Test türü**	**Tanımlama**
Birim Testi	Beyaz Kutu Testi	Birim testi, yazılım programlamasında bir tasarım ve geliştirme yöntemidir. Birim, bir bilgisayar uygulamasında test edilebilecek en küçük bölüme denir
Entegrasyon Testi	Beyaz & Kara Kutu Testi	Ayrı yazılım modüllerinin birleştirildiği ve grup olarak test edildiği yazılım testindeki aşamadır. Birim testinden sonra ve doğrulama testinden önce oluşur.
Sistem Testi	Kara Kutu Testi	Sistem testi, sistemin belirtilen şartlara uygunluğunu değerlendirmek için bütünleşik bir sistem üzerinde yapılan testtir.
Regresyon Testi	Beyaz & Kara Kutu Testi	Entegrasyon testi, bireysel yazılım modüllerinin birleştirildiği ve grup olarak test edildiği yazılım testindeki aşamadır.
Kabul Testi	Kara Kutu Testi	kabul testi, bir şartname veya sözleşmenin gerekliliklerinin karşılanıp karşılanmadığını belirlemek için yapılan bir testtir.

Çizelge 2.2 Beyaz kutu test tekniklerin kısa bir tanimlaması

Adı	Tanımlama
Temel Yol Testi	Yazılım mühendisliğinde, temel yol testi veya yapısal test, test senaryolarını tasarlamak için beyaz kutu yöntemidir. Metot, bir programın kontrol akış grafiğini analiz eder ve bir dizi doğrusal olarak bağımsız uygulama yolu bulur.
Döngü Testi	Döngü testi, tamamen döngü yapılarının geçerliliğine odaklanan bir Beyaz kutu testidir. Programdaki döngüleri test etmek için kullanılır
Kontrol-akış Testi	Kontrol akışı testi, programın kontrol akışını model olarak kullanan yapısal bir test stratejisidir

2.2 Test Vakası Önceliklendirme

Bazı kriterlere göre, daha düşük önceliğe sahip test vakalarından önce yüksek önceliğe sahip olan test vakalarını çalıştırarak test durumlarını sıralayan bir süreçtir [Gregg Rothermel vd. 2001]. Test yapana maksimum fayda sağlamak amacıyla, test durumu önceliklendirmesi, test için optimum test senaryoların sırasını keşfetmeye çalışır [S. Siz ve M. Harman. 2012]. Bu yaklaşım, regresyon testlerinde yaygın olarak kullanılır ve ilk önce [Wong vd. 1998]' tarafından tanıtılmıştır.

Test vakası önceliklendirilmesinin bazı hedefleri aşağıda belirtilmiştir:

- Test vakalarının erken hata tespit etme olasılığını artırmak
- Test sürecinde, yüksek riskli arızayı erken tespit etme oranını arttırmak
- Test işleminde, belirli kod değişiklikleriyle ilgili hataları ereken belirleme olasılığını artırmak
- Test sürecinde, bir kod kapsamı ölçütüne uymak için temin edilebilen kodun erken kapsama olasılığını artırmak
- Test yapanın, test edilmekte olan sistemin güvenilirliğine olan güvenini artırmak

Çizelge 2.3 Arıza tespit bilgisi ile test kümesinin örneği, [Elbaum vd. 2000]'dan alındı

	Test vakaları tarafından tespit edilen hatalar									
Test vakaları	1	2	3	4	5	6	7	8	9	10
A	X				X					
B	X				X	X	X			
C	X	X	X	X	X	X	X			
D					X					
E								X	X	X

Çizelge 2.2' deki örnekte, öncelikle C ve ardından E test vakasının çalıştırılması açıkça faydalıdır.

2.2.1 Test vakası önceliklendirme teknikleri

Test durumu önceliklendirme teknikleri odaklarına veya işlevselliklerine göre üç farklı kategoride gruplandırılabilir:

- Karşılaştırma grubu
- İfade seviyesi grubu
- Fonksiyon seviyesi grubu

Bu araştırmada uygulanan ikinci kategori aşağıda kısaca açıklanmıştır.

İfade seviyesi grubu: bu ayrıca, toplam ifade kapsamı önceliklendirmesi, ilave ifade kapsamı önceliklendirmesi ve toplam blok kapsamı önceliklendirmesi olarak sınıflandırılabilir. Bir programın, bir test vakası ile hangi ifadesinin kapsandığı, program enstrümantasyonu kullanılarak belirlenebilir. Böylelikle test vakaları, elde edilen teminat sırasına göre sıralanarak kapsayacakları toplam ifade sayısına göre önceliklendirilebilir ve bu sürece toplam ifade kapsamı önceliklendirmesi adı verilir. Aynı sayıda ifadenin birden fazla test vakası tarafından kapsanması durumunda test vakaları rastgele sıralanır.

Ek ifade kapsamı önceliklendirmesinde, test vakaları ulaşılan kapsama göre da sıralanır, ancak, ele alınacak bazı ifadeler olabilir ki ulaşılan kapsam hakkındaki geri bildirimin yardımı ile henüz kapsanmayan ifadelere ulaşmaya çalışır. En geniş kapsamı sağlayan test vakaları seçilmekte, daha sonra kalan tüm test vakaları ile ilgili kapsam veriler henüz ulaşılamayan ifadelere kapsamlarını gösterecek şekilde değiştirilmektedir ve bu işlem tüm ifadeler en az bir test vakasının kapsamına girinceye kadar tekrarlanmaktadır.

Toplam blok kapsam önceliği, toplam ifade kapsamı gibidir ancak test kapsamı, toplam ifadeler yerine kapsanan program bloklarına göre ölçülür. Blok kapsamı, olası her genel durum sonucunun öngörülmesinde kapsam olarak tanımlanır.

Karşılaştırma grubu

Bu kategori rastgele sıralama ve en uygun sıralama olmak üzere iki teknikten oluşur. Rastgele sıralama'da test vakalar rastgele önceliklendirilerek sıralanır. En uygun sıralama tekniğin ana amacı, bir test takımının arıza oranını artıran test vakaları keşfedilip sıralanmaktır.

Fonksiyon seviyesi grubu

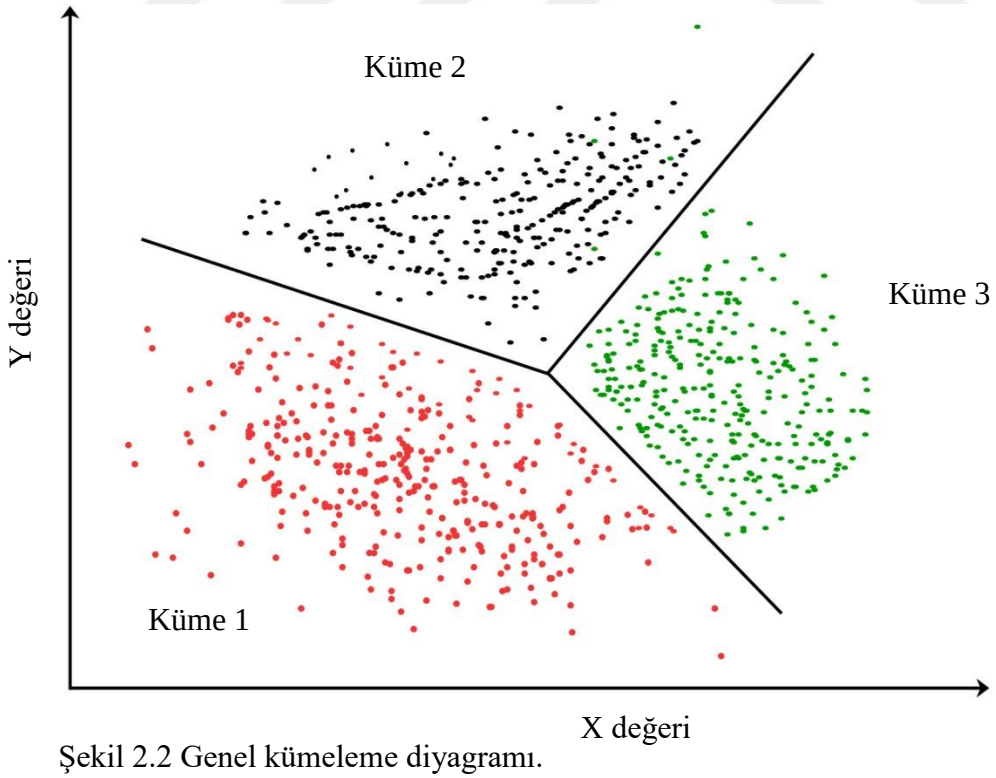
Bu kategoride yer alan tekniklerde test vakaları kapsadıkları fonksiyon sayısına göre sıralanarak önceliklendirilir. Örneğin, toplam fonksiyon kapsamı önceliklendirme teknikte test vakalar toplam kapsadıkları fonksıyon sayısına göre öncelik verilmektedir.

Çizelge 2.4 İfade düzeyinde test önceliklendirme teknikleri

Hatırlatıcı İpucu	**Açıklama**
Stmt-total	İfadelerin kapsamı sırasına göre öncelik verme
Stmt-addtl	Henüz kapsanmayan ifadelerin kapsamı sırasına göre öncelik verme
Branch-total	Blokların kapsam sırasına göre önceliklendirme

2.3 Kümeleme

Kümeleme, veri nesnelerini farklı kümelere ayırma işlemidir. Makine öğrenmesi, veri analizi ve diğer alanlarda yaygın olarak kullanılan denetimsiz bir öğrenme tekniği yöntemidir. Aynı küme içindeki veri nesneleri koleksiyonu birbirine benzer ve diğer kümelerdeki nesnelere benzemez. İyi bir kümelenme yönteminin, sınıf içi benzerliğin yüksek ve sınıflar arası benzerliğin düşük olduğu yüksek kaliteli kümeler ürettiği söylenir. Metot ve uygulaması tarafından kullanılan benzerlik ölçüsü, kümelenmenin kalitesini etkileyebilecek önemli bir faktördür.



Şekil 2.2 Genel kümeleme diyagramı.

Şekil 2.2, kümelemenin arkasındaki genel fikri göstermektedir. Her kümenin üyeleri ortak özelliklerine göre gruplandırılır, böylece kolayca ayırt edilebilirler.

Genel olarak kümeleme yöntemi şu şekilde sınıflandırılabilir: Bölümleme yöntemi ve Hiyerarşik yöntem. Bölümleme yöntemi, her bölümün bir kümeyi temsil ettiği K bölümlerine ayrılmayı amaçlar. K-means kümeleme algoritması bu kategorinin bir örneğidir. Hiyerarşik yöntem, nesnelerin veri kümesini bir grubun hiyerarşisine bölmeye çalışır. Aglomeratif kümeleme tekniği hiyerarşik tabanlı bir algoritmadır.

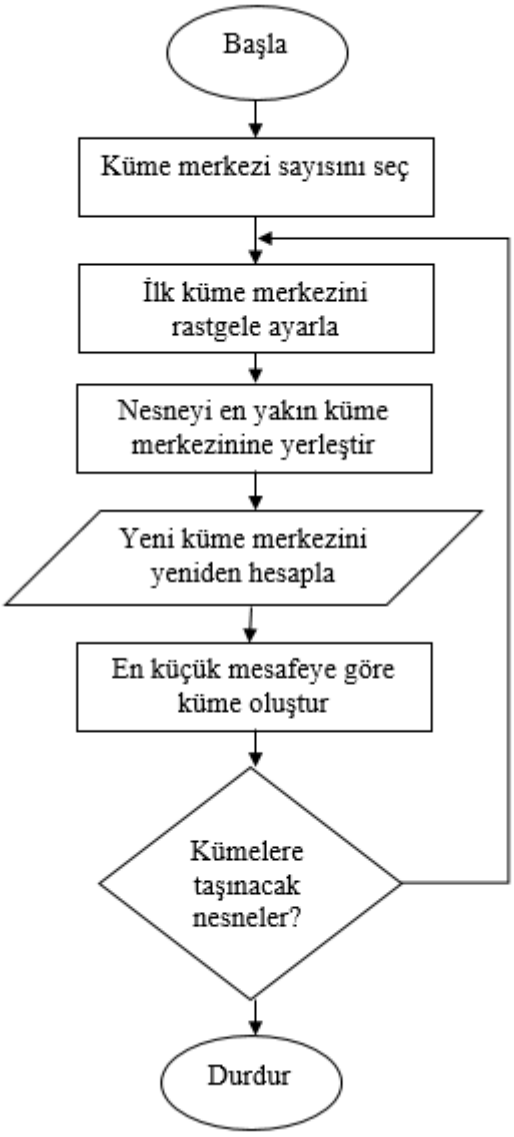
Aşağıda, bu araştırmada uygulanan K-means kümeleme algoritmasının hakkinda kısa bir açıklaması verilmektedir.

2.3.1 K-means kümeleme algoritması

En iyi bilinen bölümleme temelli kümeleme algoritmalarında biridir. Kümenin benzerliği, kümenin ağırlık merkezi olarak bilinen bir kümedeki veri nesnelerinin ortalama değeri kullanılarak hesaplanır.

Sadelik, K-means kümeleme algoritmasının [X. Wu, vd. 2007] ana avantajlarından biri olduğu söylenir, öyle ki rastgele seçilen bir başlangıç merkeziyle başlar, her girdi noktasını en yakın merkezine tahsis eder ve sonra verilen noktayı yeniden hesaplar. Bu algoritma büyük ölçüde ilk küme seçimine bağlıdır (k değeri) [G. Fung. 2001].

Her biri bir küme ortalamasını veya merkezini temsil eden nesnelerin K'sini seçmeye rasgele çalışır. Sonra nesne ve küme ortalaması arasındaki mesafeye bağlı olarak, kalan nesnelerin her biri en benzer kümeye tahsis edilir. Her küme için yeni bir ortalama hesaplanır ve bu işlem bir kriter işlevi yerine getirilinceye kadar tekrarlanır.



Şekil 2.3 K-means kümeleme algoritmasının akış şeması

2.3.2 Hiyerarşik kümeleme

Veri nesnelerini ağaç veya küme hiyerarşisi içinde gruplamayı amaçlar. Kümeleme algoritmasına dayalı bir bağlantı yaklaşımıdır. Bu teknikte kümelemek için, veri uzaklık matrisi kriterleri kullanılır ve kümeler adım adım oluşturulmaktadır [Amandeep Kaur Mann ve Navneet Kaur 2013].

Dendrogram, hiyerarşik kümelemede kümelerin hiyerarşisini temsil eden bir ağaçtır. Ağacın yaprakları bireysel veri nesnelerini temsil ederken, boş olmayan kümeler iç nesnelerini göstermektedir. Kardeş nesneler, ortak ebeveynlerinin tarafından kapsanan

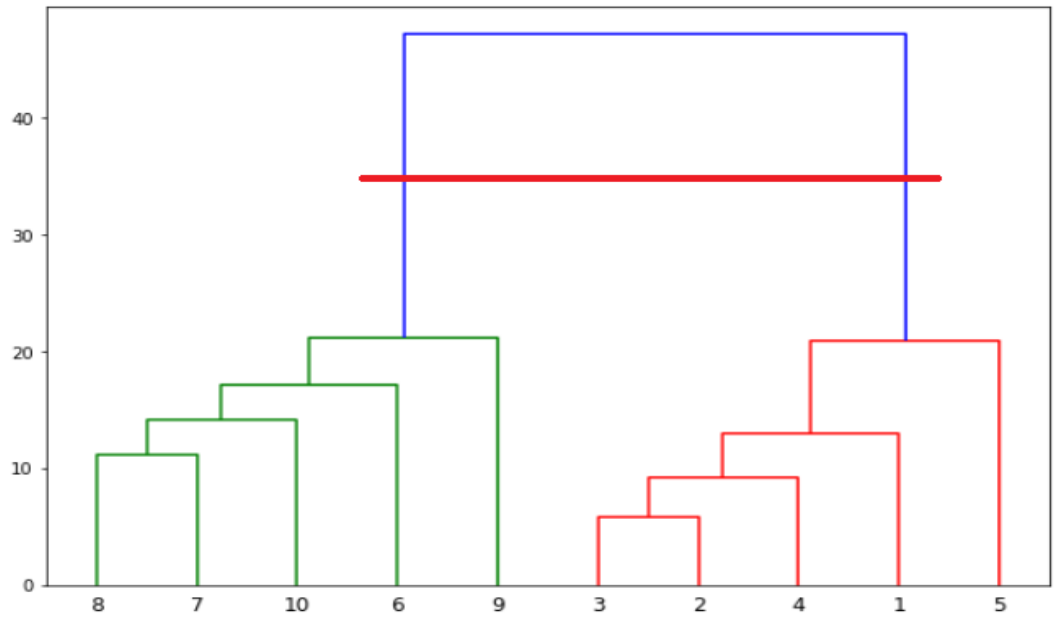
noktalarını böler ve bu veriyi farklı taneciklik konsantrasyonlarında keşfedilmeyi kolay hale getirir [Berkhin P.P 2006].

Hiyerarşik kümeleme genellikle iki kategoriye ayrılır.

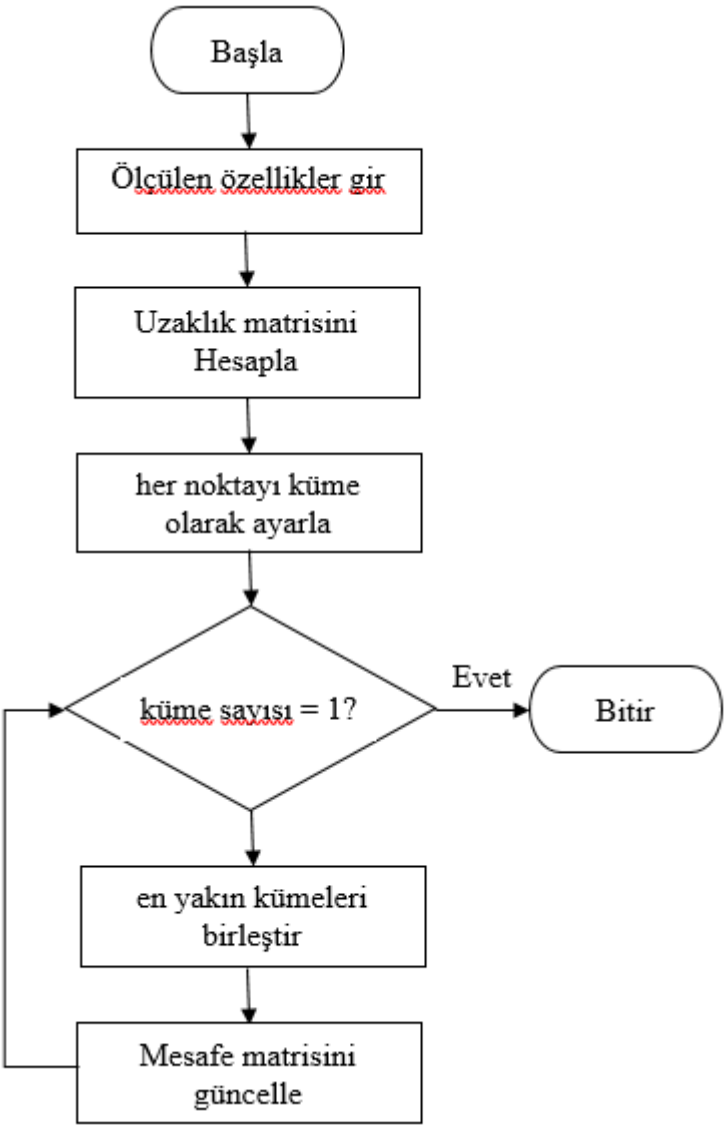
- Hiyerarşik aglomeratif kümeleme
- Hiyerarşik bölünme kümeleme

Hiyerarşik aglomeratif kümeleme

Veri nesnelerinin kümelemesinin bir noktadan başlatıldığı ve ardından düğümlerin maksimum benzerliklerine göre birleştirildiği aşağıdan yukarıya bir yaklaşımdır. Düğümleri birleştirme ve küme hiyerarşisi oluşturma süreci, hiyerarşinin tepesinde tüm düğümleri içeren tek bir küme elde edilinceye kadar devam eder [Saroj vd. 2015].



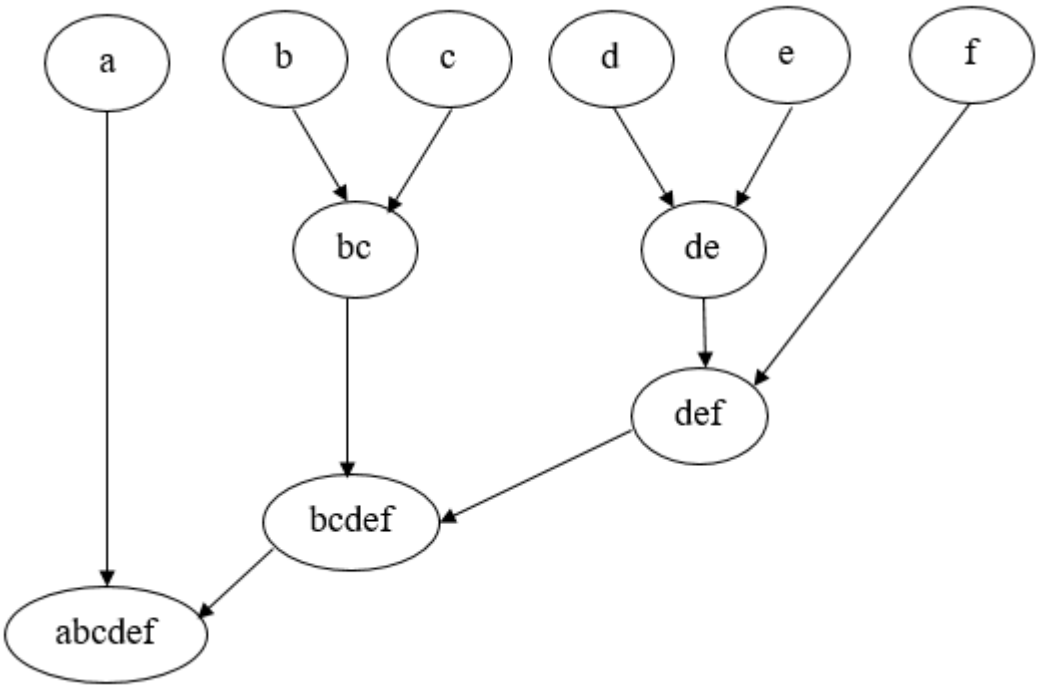
Şekil 2.4 Hiyerarşik aglomeratif kümeleme algoritması (https://stackabuse.com 2019e)



Şekil 2.5 Hiyerarşik aglomeratif kümeleme akış şeması

Hiyerarşik bölünme kümelemesi

Veri nesnelerinin kümelenmesinin tüm veri noktalarını içeren tek bir küme ile başlatıldığı ve daha sonra bu kümenin tekrar tekrar uygun bir alt kümeye bölündüğü yukarıdan aşağıya bir yaklaşımdır. Bölünme süreci, bir sonlandırma kriteri karşılanıncaya kadar devam eder [Saroj vd. 2015].



Şekil 2.6 Hiyerarşik bölünme kümeleme algoritması

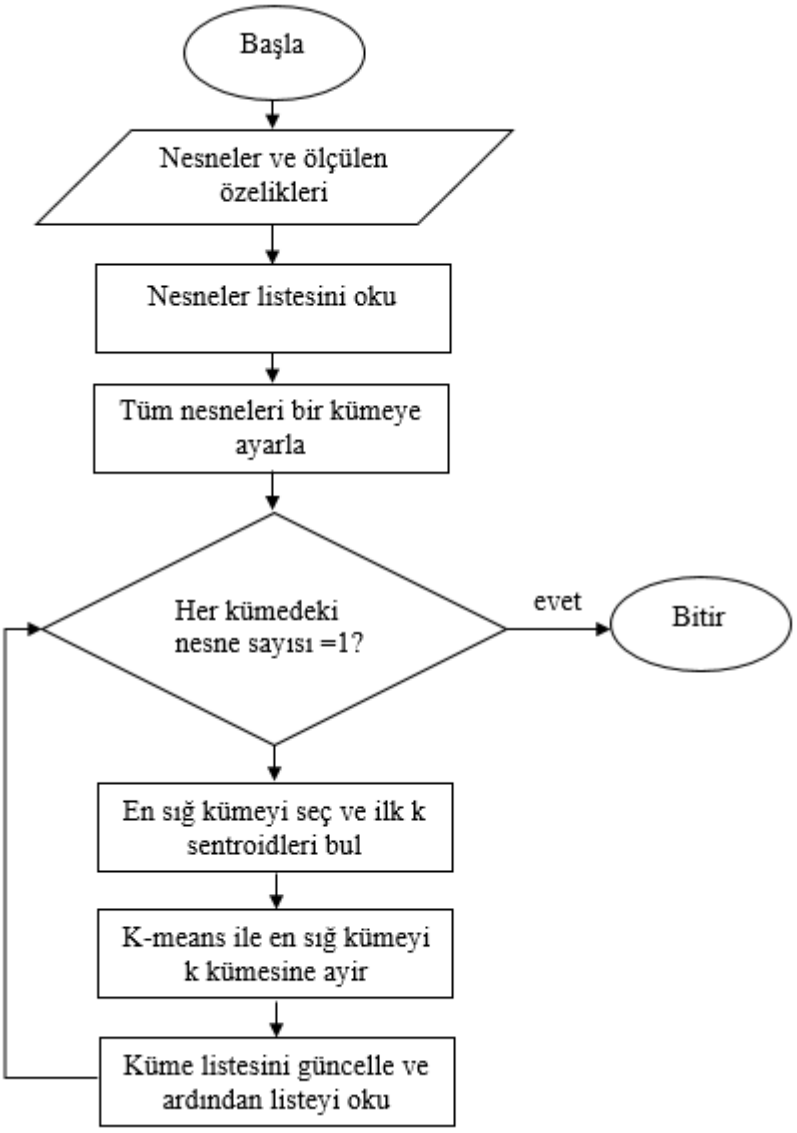
Uzaklık ölçüsü hiyerarşik kümelemede aşağıda gösterildiği gibi farklı şekillerde hesaplanmaktadır:

Tek Bağlantı Kümelemesi: İki kümedeki en yakın veri noktaları arasındaki uzaklık kullanılarak en yakın mesafeye sahip kümelerin birleştirildiği bir yöntemdir.

Tam Bağlantı Kümelemesi: İki kümedeki en uzak veri noktaları arasındaki mesafedir, tek bağlantı kümelemesinin tam tersidir. Bu yöntemde, kümeler minimum uzak mesafelerine göre birleştirilir.

Ortalama Bağlantı Kümelemesi: Burada kümeler, bir kümedeki veri noktalarının başka bir kümedeki veri noktaların en düşük ortalama mesafesine göre birleştirilir.

Centroid Bağlantı Kümelemesi: Bu yöntemde öncelikle her bir kümenin ortalaması hesaplanır. Bundan sonar kümelerin centroidleri arasındaki mesafe hesaplanır ve daha sonra kümeler en düşük ortalama mesafeye göre birleştirilir.



Şekil 2.7 Hiyerarşik bölünme kümeleme akış şeması

Hiyerarşik kümeleme algoritmasının avantajları

- Ayrıntı seviyesine göre gömülü esneklik
- Herhangi bir benzerlik veya mesafeyi kolayca idare edebilme
- Herhangi bir özellik türüne uygulanabilir

Hiyerarşik kümeleme algoritmasının dezavantajları

- Doğru bir sonlandırma kriteri seçmek zor
- Çoğu hiyerarşik algoritma, geliştirme amacıyla oluşturulduktan sonra kümeleri tekrar ziyaret etmez

2.4 İlgili Çalışmalar

Mutasyon testinin hesaplama probleminin üstesinden gelmek için, çeşitli maliyet azaltma teknikleri önerilmiştir. [A. J. Offutt ve R. H. Untch, 2001] Anket çalışmasına göre, maliyet düşürme yöntemleri üç kategoriye ayrılır: "daha azını yapın", "daha hızlı yapın" ve "daha akıllı yapın". Bu tez yalnızca üretilen mutantların azaltılmasına odaklanmaktadır (daha azını yapmak). Literatürde birçok yöntem, üretilen mutantların azaltılması ile ilgilidir ancak [Jia Y ve Harman M, 2018] 'de gösterildiği gibi, en çok çalışılan teknik seçici mutasyondur.

2.4.1 Mutasyon test maliyetini düşürme ile ilgili çalışmalar

Seçici mutasyon:

Seçici mutasyonun arkasındaki ana fikir, kendisine uygulanan mutasyon operatörlerinin sayısının azaltılmasıyla mutantların sayısında bir azalmanın elde edilebileceğidir. Seçici mutasyon, önemli bir test yeterliliği kaybı olmadan, onu üreten küçük bir mutasyon operatörü seti bularak, olası tüm mutantların bir alt kümesini seçmeye çalışır. Bu yöntem ilk önce Mathur [A.P. Mathur, 1991] tarafından tanıtılmıştır ve daha sonra Offutt vd. [A.J. Offutt, G. Rothermel, and C. Zapf, 1993] tarafından genişletilmiştir. Bazı mutasyon operatörlerinin diğerlerinden daha fazla mutant oluşturduğu söylenir. Örneğin, [K.N. King ve A.J. Offutt, 1991] çalışmasında, 22 Mothra operatörünün iki mutasyon operatörünün (ASR ve SVR) tüm mutantların yaklaşık %30 veya %40'ını oluşturduğu tarif edilmiştir. Offutt vd. [A.J. Offutt, G. Rothermel ve C. Zapf, 1993]'ye göre, iki mutasyon operatörünü çıkartarak, mutantların %24'ünün düşürüldüğü ortalama % 99.99'luk bir mutasyon skoru elde edilebilir. Aynı çalışmada, 4 mutasyon operatörünün ihmal edilmesinin, % 41'lik mutant azaltmasıyla % 99,84'lük bir mutasyon skoru elde edildiği de bildirilmiştir. Wong'un bu çalışmalarında [W.E. Wong, 1993] ve Mathur [W.E. Wong ve A.P. Mathur, 1995] test etkinliğine dayalı seçim adı verilen başka bir seçim tekniği uygulanmaktadır. Bu çalışmalarda sadece iki mutasyon operatörü kullanmak: ABS ve ROR önerilmişti. Deneysel sonuçlar, bu iki mutasyon operatörünün pratikte mutasyon skoruının yüzde 5 azalmasıyla yüzde 80'lik bir mutant azalması elde ettiğini göstermektedir.

Offutt vd. [A.J. Offutt, A. Lee, G. Rothermel, R.H. Untch ve C. Zapf, 1996] aynı seçim stratejisini kullanarak daha önceki N-seçici mutasyon çalışmalarını genişletti. Mothra

mutasyon operatörlerine göre, bunları üç kategoride gruplandırdılar: operandlar, ifadeler ve ifadeler. Her sınıftan operatörleri atlarlar, operandlardan ve ifadelerden olan bu beş operatörün: (ABS, UOI, LCR, ROR ve AOR) % 99,5 mutasyon skoruı alarak anahtar operatörleri olduğunu bulmuştular.

Bottaci ve Mresa'nın çalışmasında bir diğer seçici mutasyon türü de tanıtılmıştır [E.S. Mresa ve L. Bottaci, 1999]. Bu çalışmada eşdeğer mutantların tespit maliyeti göz önünde bulundurulmuştur ve sonuçlar, etkinliği korurken eşdeğer mutantların azaltılması olasılığını gösterdi. Öte yandan, Namin ve Andrews bu çalışmalarında [A.S. Namin ve J.H. Andrews, 2006], [A.S. Namin ve J.H. Andrews, 2007], [A.S. Namin, J.H. Andrews ve D.J. Murdoch, 2008], istatistiksel bir problem: değişken seçimi veya azaltma problem olarak ifade ederek farklı tipte bir seçici mutasyon ortaya çıkardılar. 108 C mutasyon operatörlerinden 28 mutasyon operatörünün bir alt kümesini belirlemek için doğrusal istatistiksel yaklaşımlar uyguladılar. Üretilen tüm mutantların% 92'sinin bu operatörler tarafından düşürüldüğünü, ayrıca bu 28 operatörün bir test vakalarının etkinliğini tahmin etmek için yeterli olduğunu iddia etmişlerdir.

Mümkün olan tüm mutasyon operatörlerinden yeterli bir mutasyon operatörü kümesinin seçilmesi için bir kılavuz [H. Agrawal et al. 1989]'de tanımlanmıştır. Daha sonra kılavuz Proteum'un 77 C mutasyon operatörlerine uygulandı [E. F. Barbosa, J. C. Maldonado, and A. M. R. Vincenzi, 2001]. % 99.6'lık bir mutasyon skoru elde eden ve% 65'inde mutant azalma sağlayan 10 seçilmiş mutasyon operatörü seti elde edildi.

Mutant örnekleme:

Mutant örnekleme adı verilen başka bir mutant azaltma yaklaşımı vardır. Bu, tüm mutant grubundan küçük bir mutant alt grubunu rastgele seçen bir yöntemden başka bir şey değildir. Bu araştırmalarda belirtildiği gibi [T.A. Budd, 1980] ve [A.T. Acree, 1980] tüm olası mutantlar ilk önce üretilir, daha sonra mutasyon analizi için bu mutantların % x'i rastgele seçilirken geri kalanı yoksayılır. Wong ve Mathur çalışmalarında yapılan deneylere göre [W.E. Wong, 1993], [A.P. Mathur ve W.E. Wong, 1993], % 10'dan % 40'a kadar % x oranında rastgele bir seçim oranı kullanılarak, mutantların % 10'dan daha yüksek bir % x değerle geçerli olduğunu ima etti.

Mutasyon kümelenmesi:

Rastgele mutant seçmeye bir alternatif olarak, kümeleme algoritmasının yardımı ile bir mutant alt kümesini seçen mutasyon kümelenmesi olarak bilinen başka bir yöntem vardır. Bu teknik ilk olarak S. Hussain [S. Hussain, 2008] tarafından önerilmiştir. Temel fikir, tüm birinci mertebe mutantlarının üretilmesi, ardından birinci mertebe mutantlarının, katlanabilir test durumlarına göre farklı kümelenme gruplarına sınıflandırılması için bir kümeleme algoritması uygulanmasıdır. Her bir kümedeki mutantların aynı test vakaları tarafından öldürülme garantisi vardır. Daha sonra her bir kümeden, mutasyon testinde kullanılmak üzere az sayıda mutant seçilirken diğer kalanlar dikkate alınmaz.

Yüksek Mertebeden Mutasyon

Mutantlar iki yolla üretilebilir, sadece bir kez mutasyon operatörleri uygulanarak: birinci dereceden mutantlar (FOMs) veya birden fazla uygulayarak: daha yüksek dereceli mutantlar (HOMs). Yüksek Mertebeden Mutasyon, Jia ve Harman [Y. Jia ve M. Harman, 2008] tarafından önerilen bir mutasyon testidir. Bu çalışmada, öldürülmesi daha zor olan HOM'ların, inşa edildiği FOM'lardan daha zor olduğu varsayılmıştır. Mutantların sayısını azaltmak için, FOM'lerin tek HOM ile değiştirilmesi tercih edilebilir. Polo vd. [M Polo, M. Piattini ve I. Garcia-Rodriguez, 2008] çalışmasında, ikinci derece mutantları üretmek için, birinci dereceden mutantları birleştirmek için farklı algoritmalar önerilmiştir. Deneylerinde, ikinci dereceden mutantlar uygulanarak test çabasının yaklaşık% 50'si azaltılmıştır. Langdon ve diğ. [W.B. Langdon, M. Harman ve Y. Jia, 2009], [24] daha yüksek dereceli mutantlar üretmek için çok nesne genetik programlama kullanmıştır. Deneylerinde diğer birinci derece mutantlardan öldürmesi daha zor olan gerçekçi yüksek dereceli mutantlar bulunmuştur.

Eşdeğer mutantların tespiti:

[R.M. Hieronlar, M. Harman, S. Danicic, 1999] çalışmasında, eşdeğer mutantları tespit etemek için tanımlama işlemine yardımcı olarak program dilimleme tekniği önerilmiştir. asıl amacı, birinci derece mutantların yerine, yüksek dereceden bir mutant kümesi üretmektir. yüksek mertebe mutantları kullanmak eşdeğer mutantlarla başa çıkmak için önerilen yaklaşımlardan biridir ve bu çalışmalarda [M. Kintis, M.

Papadakis, N. Malevris, 2010], [M. Papadakis, N. Malevris, 2010] bildirildiği gibi büyük ölçüde daha az eşdeğerleri üretmektedir.

Diğer maliyet düşürme teknikleri:

Pedro Delgado-Perez vd. Çalışmalarında [Delgado-Pérez, Pedro, 2018] "kapsama bilgilerinin her bir mutasyon operatörüne atanan kaliteyi nasıl etkilediği" sorusuna bir cevap verdiler. Cevaplarına göre, kapsama dayalı kalite metrikinin mutasyon operatörlerine verdiği değerler, orijinal kalite metriklerine kıyasla farklı ölçeklerdedir. Ayrıca, ulaşılması zor olan mutantların üretilmesinin, mutasyon operatörü kalitesi üzerinde önemli bir etkisi olmadığını belirtti.

Yong Liu ve diğ. [Yong Liu, Zheng Li, Ruilian Zhao, Pei Gong, 2018] çalışmalarında iki optimizasyon içeren dinamik bir mutasyon yürütme stratejisi sundular: mutasyon yürütme optimizasyonu ve test durumu yürütme mutasyonu. Bu optimizasyonlar, mutantların ve test durumlarının yürütme sırasını dinamik olarak ayarlayarak ifadelerin daha hızlı hesaplama şüphesi değerlerine odaklanır. Ampirik sonuçlar, doğruluk kaybı olmadan bu optimizasyonların mutasyona dayalı hata lokalizasyonunun maliyetini düşürdüğünü göstermektedir.

Chang-ai Sun vd. [Chang-ai Sun, Feifei Xue, Huai Liu, Xiangyu Zhang, 2017] çalışmasıda da, test edilen programdaki yol derinliği perspektifinden yeni bir mutant azaltma yaklaşımı önerdiler. Buna dayanarak, yola duyarlı iki sezgisel kural tanımladılar: modül derinliği ve döngü derinliği, ve bunları mutant azaltma stratejileri geliştirmek için ifade ve operatör tabanlı mutasyon seçimi ile birleştirdiler. Deneysel sonuçlarında Mutant azaltma stratejilerinin her ikisinin de, daha temsili mutantların seçilmesi açısından saf rastgele mutant seçim stratejisinden daha etkili ve sistematik olduğunu belirtmişlerdir.

Baskınlık ilişkisine dayanan, mutantları azaltmanın yeni bir yöntemi Dunwei Gong vd. [Dunwei Gong, Gongjie Zhang, Xiangjuan Yao, Fanlin Meng, 2017] tarafından tanıtılmıştır. Yöntemlerinde, ilk mutant dalları mutasyondan önceki ve sonraki ifadelere dayanarak oluşturulur. Bundan sonra, tüm mutant dalların orijinal programa kaynaştırılmasıyla yeni bir program oluşturulur. Daha sonra, yeni programda mutant dallar arasındaki baskınlık ilişkisi analiz edilir ve redüksiyondan sonra mutantlara

karşılık gelen dominant olmayan mutant dallar elde edilir. Deney sonuçları, yöntemlerinin ortalama% 80'den fazla mutantı azalttığını gösterirken, mutasyon testinin etkinliğini büyük ölçüde artırmaktadır.

Mutasyon testinin, hesaplama açısından problemi ile başa çıkmak için farklı teknikler ve yöntemler getirilmiş olmasına rağmen, yine de kusursuz değildir. Seçici mutasyon çalışmalarının çoğu küçük programlarda yapılmıştır ve değerlendirilmiştir. Örneğin. [Barbosa vd. 2001] toplam 619 ifadeye sahip 27 C programında deneyler yapıldı. [Namin ve ark. 2006] deneylerde 2188 ifade ile yedi C programı yürütülmüştür ve değerlendirilmiştir.

Seçici mutasyon ve mutant örneklemenin ana zayıflığı, birinci yöntemde, bir mutasyon operatörü alt grubu seçilip, bu seçilmiş operatörler kullanılarak mutantlar üretilmesi, ve bazı kritik mutasyon operatörleri göz ardı edilmektedir. ikinci yöntemde ise, mutantlar bütün mutantlar grubundan rastgele seçilir ve bazı kritik mutantlar atılır.

Mutasyon kümelenmesinde en detaylı ve en öncelikli çalışma S. Hussain [S. Hussain, 2008]'in çalışmasıdır, ancak iki önemli yönden yoksundur. Birincisi, tamamen mutasyon testinin karşılaştığı ana zorluklardan eşdeğer mutantlar olan ile ilgilenmemektedir. İkincisi ise, küçük ve orta ölçekli programlar üzerinde deneyler yapılması ve değerlendirilmesidir.

2.4.2 Test vakası önceliklendirme ile ilgili çalışmalar

[Rothermel vd. 2000]'ye göre, erken teşhis kapsamı maksimize edildiğinde, arıza tespitinde erken maksimize etme şansı arttırılabilir. Aynı çalışmada, farklı arıza tespit oranı ikameleriyle aynı algoritmaya uygulanan farklı önceliklendirme tekniklerinin ampirik bir çalışmasının raporu edilmiştir. İfade toplamı, İfade eki ve blok toplamı dikkate alınan vekiller arasındaydı.

[Elbaum vd. 2001], [Rothermel vd. 2000]'in çalışmalarını çenişletirerek İşlev kapsamı da dahil olmak üzere yeni test önceliklendirme teknikleri tanıtılmıştır. Bu çalışmada, kaba tanecikli yaklaşımlarla daha düşük ortalama hata tespit yüzdesinin üretileceği varsayılmıştır, ve istatistiksel olarak da doğrulanmıştır.

Srivastava, A ve Thiagarajan J [Srivastava A, ve Thiagarajan J 2002] çalışmasında açgözlü tabanlı bir önceliklendirme yaklaşımı kullanılmıştır. Test edilen sistemin önceki

sürümünün ikili kodunu yeni sürümüyle karşılaştırarak değiştirilmiş kod blokları tanımlandı. Ardından açgözlü tabanlı önceliklendirme kullanılarak değiştirilmiş kod bloklarının kapsamına göre test durumlarına öncelik verilmiştir.

JUnit test ortamına kapsama dayalı önceliklendirme teknikleri uygulanmıştır [Do H, Rothermel G ve Kinneer A. 2004]. Bu çalışmanın sonuçlarında gösterildiği gibi, JUnit test vakalarının uygulanmasına öncelik verilerek geliştirilmiş hata tespit oranı elde edilmiştir. Ayrıca, derecelendirilmemiş sıralamaya göre daha yüksek ortalama hata saptama yüzdesi değerinin bazen rastgele önceliklendirme ile elde edilebileceği belirtilmiştir.

[Li vd., 2007], Rastgele önceliklendirme, tepe tırmanma algoritması, genetik algoritma ve açgözlü tabanlı algoritmaların karşılaştırmalı bir çalışmasını önermiştir. Test vakası önceliği çeşitli meta-sezgisel taramalara uygulandı. Program Space ve Siemens paketi programları üzerinde deneyler yapıldı, ortalama hata tespit yüzdesi yerine, deney programları ortalama blok kapsamı yüzdesine göre değerlendirildi. Ek açgözlü algoritma, ampirik sonuçlarda belirtildiği gibi, genel olarak en verimli algoritma seçilmiş.

Dağıtım temelli yaklaşım adı verilen bir başka önceliklendirme tekniği [Leon ve Podgurski. 2003]' tarafından önerildi. Bu teknikte test durumlarına çok boyutlu profil alanında profil dağılımı açısından öncelik verilir ve en aza indirilir. farklılık ölçüsü, iki girdi profili arasındaki gerçek derece farklılığı temsil eden bir sayı üreten işlev, test durumu profilleri oluşturur. Test durumları, bu ölçü kullanılarak benzerliklerine göre kümelenebilir. Bu çalışmaya göre, test durumları bu şekilde kümelenerek aşağıdaki bilgiler elde edilebilir.

- Bir grup gereksiz test vakası, benzer profillerin kümeleriyle gösterilebilir
- Başarısızlığa neden olma olasılığı daha fazla olduğu düşünülen koşullara neden olan test durumları izole kümelerde bulunabilir
- Olağandışı kullanım davranışları düşük yoğunluklu profil alanın alanları ile gösterilebilir

3. MATERYAL VE YÖNTEM

Bu bölümde ilk olarak mutasyon testi maliyetini azaltmak için önerilen yöntem hakkında bilgi verilmektedir. Daha sonra önerilen yöntemde yapılan çalışmada kullanılan veri ve programlar anlatılmaktadır. Son olarak bu veri ve programların yapılan deneylerde nasıl kullandığı vurgulanmaktadır.

3.1 Mutasyon Test Maliyetini Azaltma Yöntemi

İkinci bölümde belirtilen problemleri göz önünde bulundurarak, bu tez eşdeğer mutantlarla başa çıkmak üzere aynı anda çok sayıda mutantı azaltmanın bir yolunu sunmaktadır. İndirgeme yaklaşımı üç temel aşamadan oluşur: eşdeğer mutant tespiti, test vakası önceliği ve mutasyon kümelenmesi.

Bir mutantın orijinal programla aynı olup olmadığına karar vermek, mutasyon skorunu hesaplamadan önce önemli bir adımdır, ancak aynı zamanda iyi bilinen çözülemez problem ve açık bir araştırma sorunudur. Genel olarak, denklik problem için bir algoritmik çözüm bulunmamaktadır [Offutt, A. J. and W. M. Craft.1994]. Sezgisel olarak, yapılan varsayım bir mutantın en az bir test vakasıyla tespit edilebildiği takdirde eşdeğer bir mutant olma ihtimalinin daha düşük olmasıdır.

Literatürde bir basit hata içeren mutantlara birinci derece mutantlar, iki basit hata içeren mutantlara ise ikinci derece mutantlar denir [M. Polo vd. 2009]. İkinci derece mutantların üretilmesi, eşdeğer mutantları tespit etmek için kullanılan en iyi yöntemlerden biridir. Bu fikre dayanarak, Polo vd. [M. Polo vd. 2009], ikinci dereceden mutantlar (SOMs) üretmek ve birinci derece mutantları (FOMs) birleştirmek için üç farklı algoritma (Last2First, DifferentOperators ve RandomMix) tanıtmıştır.

Last2First, bu çalışmada, önerilen yöntemin ilk aşamasında eşdeğer mutantları tespit etmek için kullanılmıştır. Üretilen birinci mertebe mutantlarının listesini alır. İlk mutantı sonuncuyla birleştirmeye çalışır, sonra ikinciyi sonrakine vb. Şeklinde devam eder. Last2First'in eşdeğer mutantları tespit etmek için önerilen diğer yöntemlere göre daha iyi performans gösterdiğinden bahsetmek gerekir ki, aynı zamanda Papadakis ve Malevris' in [M. Papadakis and N. Malevris. 2010] çalışmalarında da denenmiş ve daha fazla araştırılmıştır.

```
Algorithm  Last2First(program, operators[ ])
1: LET firstOrderMutants be an empty list
2: FOR ALL operator in operators
3:      mutationPoints[] ← operator.countMutantionPoints(program)
4:      FOR ALL point in mutationPoints
5:         possibleMutation [] = operator.count PossibleMutations(program,point)
6:         FOR EACH possibleMutant in possibleMutations
           DO
7:            newMutant ← operator.mutate(program, point, possibleMutant)
8:           firstOrderMutants ← newMutant
9:         END FOR
10:    END FOR
11: END FOR
12: LET secondOrderMutants be an empty list
13: WHILE firstOrderMutants.size > 1 DO
14:     fom1 ← firstOrderMutants → last
15:     firstOrderMutants.remove(fom1)
16:    form2 ← firstOrderMutants → first
17:    IF firstOrderMutants.size ≠ 2 THEN
18:       firstOrderMutants.remove(form2)
19:    END IF
20:    operator ← fom2 → operator
21:    newMutant ← operator.mutate(fom1→program, form2→point,
       fom2→possibleMutants)
22:   secondOrderMutants ← newMutant
23: END WHILE
24: RETURN secondOrderMutants
```

Şekil 3.1 Eşdeğer mutantları tespit etme algoritması, (Lech Madeyski vd. 2014).

Eşdeğer mutantları tespit ettikten sonra, bu çalışmada önerilen yaklaşım test vakalarına karşı çalıştırmak için eşdeğer olmayan mutantları alır. Toplam blok kapsamı önceliklendirme tekniğini izleyerek en çok kapsayan / öldüren test vakalarına daha yüksek öncelik verir. Toplam blok kapsamı tekniği, test vakalarını kapsadıkları toplam blok sayısına (tek girdi, tek çıkış dizisi sıraları) dayanarak basitçe önceliklendiren ve bu sayıya göre sıralayan bir test vakaları önceliklendirme yöntemidir. Bu, test uygulamasında erkenden arıza tespit etme olasılığını artırır ve uygulama süresini potansiyel olarak azaltır.

Çizelge 3.1 Deneyde kullanılan verilerin örneği

	T1	T2	T3	T4	T5	T6
M1	1	1				
M2	1	1			1	
M3		1				1
M4			1			1
M5			1			1
M6			1			1

Örnek olarak, Çizelge 3.1'de altı mutant içeren bir program ve altı test vakasıyla birlikte bir test kümesini göstermektedir. Tam çalıştırma 6x6 = 36 gerektirecektir. Her mutanta karşı bir test vakası çalıştırma yerine, sadece hayatta kalan / yaşayan mutantlara karşı uygulanabilir. Örneğin, T1, M1 ve M2'yi öldürür, diğer mutantlar hayatta kalır, sonra M1 ve M2 zaten T1 tarafından kaplanıp öldürüldüğünden dolayı, tüm mutantlara karşı T2 uygulanması, zaman alıcı olduğu kadar gerekli değildir. Öncelik verildikten sonra test senaryolarının yürütme sırası şöyle olacaktır: {T6, T3, T2, T1, T5} veya {T6, T2, T3, T1, T5}. Test vakaları önceliklendirmesinden sonra, T4 uygulamasına gerek yoktur, çünkü ulaşılıp öldürülecek hiçbir mutant kalmaz.

Son olarak, üçüncü aşamada, eşdeğer mutantların tespiti ve test vakalarının önceliklendirilmesinin ardından, mutantlar, bir kümeleme algoritması uygulanarak, öldüren test vakalarına göre farklı gruplara ayrılır. Her bir kümedeki mutantların aynı test vakaları tarafından öldürülme garantisi vardır. Daha sonra, mutasyon testine devam etmek için her kümeden az sayıda mutant seçilir ve diğerleri atılır. Bu durumda bir K-means kümeleme algoritması seçilmiştir. K-means algoritması en basit ve yaygın olarak kullanılan denetimsiz bir makine öğrenme algoritmasıdır. Verilerin kümelenmesi prosedüründe de benzerlik önlemleri önemlidir. İki veri nesnesi veya kümesi arasındaki benzerlik ve farklılığa karar verirler. Mutasyon kümelemesini gerçekleştirmek için, benzerlik ölçüsü olarak Düzen uzaklığı seçilmiştir. Düzenleme mesafesi, bir sözcüğü diğeri ile değiştirmek için gereken minimum tek karakterli düzenleme sayısıdır (ekleme, silme veya değiştirme).

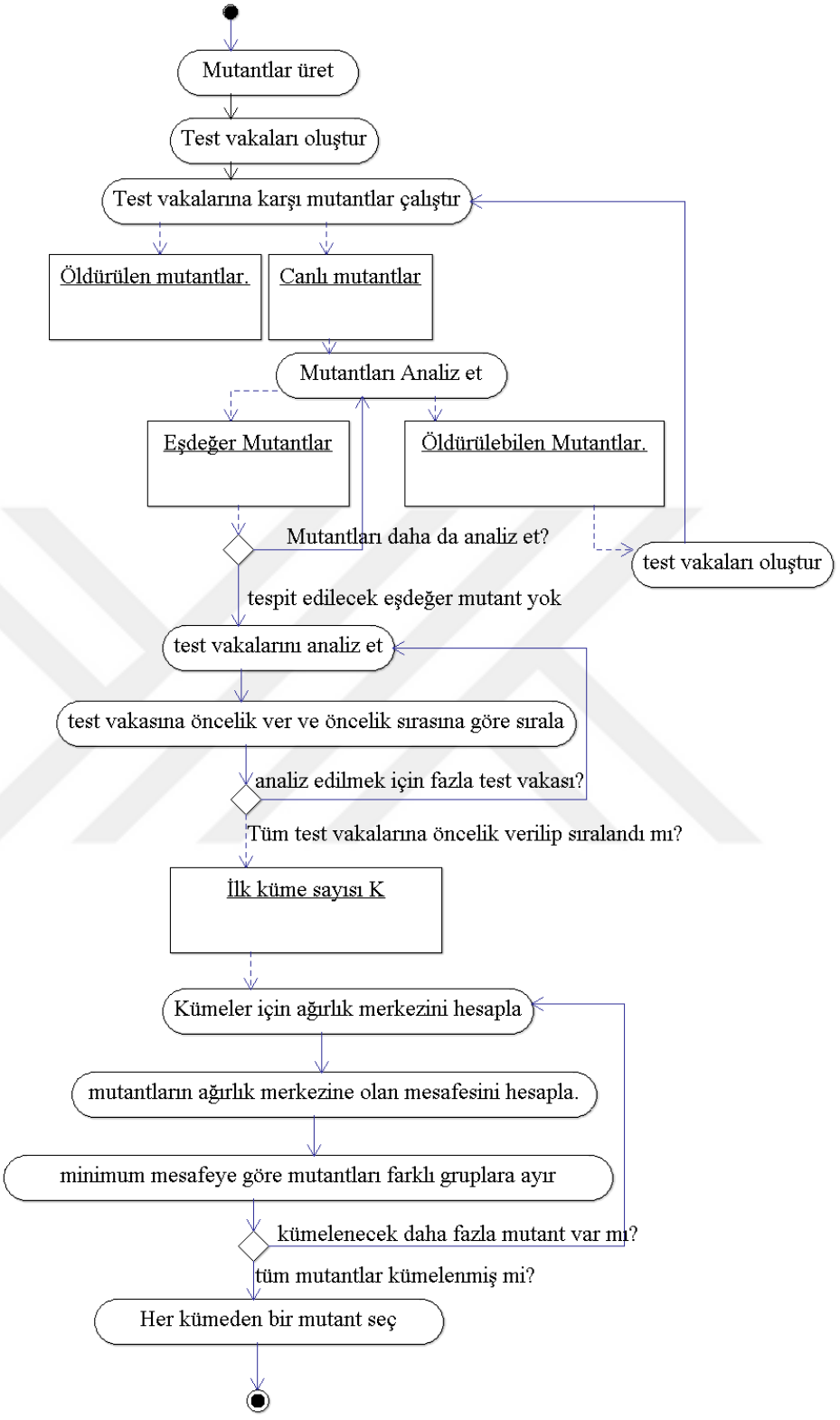
Input (S_1 , S_2)

```
1   int secondOrderMutant [i , j] = 0
2   for i ← 1 to |S1|
3   do secondOrderMutant [i , 0] = i
4   for j ← 1 to |S2|
5   do secondOrderMutant [0 , j] = j
6   for i ← 1 to |S1|
7   do for j ← 1 to |S2|
8       do secondOrderMutant [i , j] = min {secondOrderMutant [i —1, j —1] +
            if(S1[i]  = S2[j]) then 0 else 1,
9                 secondOrderMutant [i —1, j] + 1,
10                secondOrderMutant [i, j —1] +1}
11  return secondOrderMutant [|S1 |, |S2|]
```

Şekil 3.2 Düzenleme mesafesini hesaplama algoritması

K-means kümelenme algoritması, [S. Hussain. 2008]'de belirtildiği gibi böyle çalışır:

- Bir ilk K küme değeri seçin, K değeri farklı mutantların sayısından daha az olmalıdır.
- Rasgele farklı K mutantları seçin.
- Kümenin ağırlık merkezini hesaplayın.
- Düzenleme mesafesini kullanarak mutantların merkeze olan mesafesini hesaplayın.
- Mutantları, aralarındaki minimum mesafeye bağlı olarak farklı kümelere ayırın.
- Yeni kümelerin (ve eski kümelerin) merkezlerini yeniden hesaplayın.



Şekil 3.3 Yaklaşımı temsil eden faaliyet diyagramı

3.2 Deneyler

Bu kısım, önerilen yöntemin mutasyon test maliyetini etkili bir şekilde azaltıp azaltmadığını belirlemek için deneylerin nasıl yapıldığını açıklamaktadır.

Deneyin temel amacı, mutasyon kümelenmesini test vakası önceliği ile araştırmak ve değerlendirmektir, bu amaçla yedi farklı Java programının deneyi yapıldı. Programlar mutant sayısı, test vakaları ve boyut bakımından farklıdır. Test konularıyla birlikte tüm konu programları kod havuzlarında bulunmuştur (http://timeandmoney.sourceforge.net/ 2019a), (http://jtopas.sourceforge.net/jtopas/index.html 2019b), (https://sourceforge.net/projects/jdepends/ 2019c), (https://x-stream.github.io/tutorial.html 2019e), (https://sourceforge.net/projects/jaxen/ 2019f), (https://commons.apache.org/proper/commons-lang/ 2019g). Bu programlar, Lingming Zhang ve arkadaşlarının [Zhang, Lingming vd. 2013] yüksek referanslı çalışmalarında deney için kullanıldığından seçilmiştir. Bu programların büyüklüğü 46 ile 15,855 kod satırı arasında değişmektedir. Herhangi bir test vakası tarafından öldürülmeyen mutantlar, deneyleri ikinci aşamada daha da sürdürmek için yöntemin birinci aşamasında çıkarılır. Deneylerde kullanılan veriler 0s ve 1s'den oluşmaktadır. Mutant herhangi bir test vakası tarafından öldürülürse 1, aksi takdirde 0 olarak temsil edilir. Çizelge 3.2, uygulanan yaklaşımdan önce söz konusu programların saf mutasyon testinin sonucunu özetlemektedir. Çizelgede gösterildiği gibi mutasyon skoru eşdeğer mutantlardan dolayı çok düşüktür.

Çizelge 3.2 Yöntem kullanılmadığında konu programlarının sonuçlarının özeti.

Programı	Uygulama vakti	Mutasyon skoru
Triangle	35 saniye	55%
Time & Money	3 dakika	40%
JDepend	3.5 dakika	35%
JTopas	2 dakika	60%
XStream	9.15 dakika	65%
Jaxen	8.45 dakika	62%
Commons-lang	8.55 dakika	70%

Çizelge 3.3 Konu programları ve özelikleri.

	Programı	LOC	Mutant Sayısı	TC sayısı
1	Triangle	46	216	18
2	Jdepend	7338	2725	60
3	Time & Money	7566	3220	227
4	Jtopas	2775	1753	74
5	XStream	15,855	8,050	340
6	Jaxen	11,062	7,125	293
7	Commons-Lang	14158	7,520	375

Deneyleri devam ettirmek için veri önemlidir, veri elde etmek için EClipse: bir Java mutasyon test aracı kullanılarak mutantlar üretilmiştir. MuClipse [B. H. Smith and L. Williams 2007], mevcut MuJava mutasyon motoru ve Eclipse IDE arasında köprü sağlayan bir Eclipse eklentisidir. Bu tezde yapılan çalışmada mutantlar üretmek için kullanılan mutasyon operatörler, MuClipse mutasyon test araçlarında mevcut olduğu kadar, en çok kullanılan java mutasyon operatörleri ve mutantların çoğunun üretilmesinden sorumlu olanlardır. Çizelge 1.3'e bakınız.

Aşağıda, deneyde kullanılan programların kısa bir açıklaması verilmiştir. Deneyde kullanılan konu programların tüm test vakaları oluşturulmamıştır. Hepsi konu programlarıyla kod depolarında bulunmuştur.

Triangle: Deneyi gerçekleştirmek için seçilen ilk program Üçgen programıdır. Mutasyon test alanında basit ve çok kullanılan bir programdır. Üçgenin tipini, üç tarafın verilen uzunluğundan belirlemeye çalışır. 46 kod satırı, 217 mutant ve 14 test vakası vardır.

Time/money: Deneyleri yapmak için seçilen ikinci programdır. Bu programın arkasındaki ana fikir zaman, para gibi tekrarlayan alanlarda temel kavramları korumak için kod geliştirmektir. Mutasyon test araştırmalarında çok kullanıldığından dolayı seçilmiştir. Zaman ve parayı manipüle eden bir dizi sınıfa sahiptir. Bu programın 7566 kod satırı ve 227 test vakası vardır**.**

Jdepend: Jdepend, denemenin üçüncü programıdır. Java sınıfını ve kaynak dosya dizinlerini geçer ve her bir java paketi için tasarım kalitesi ölçümlerini oluşturur. Ayrıca, paket bağımlılıklarını etkin bir şekilde yönetmek ve kontrol etmek için genişletilebilirliği, tekrar kullanılabilirliği ve bakımı açısından bir tasarımın kalitesini otomatik olarak sağlar. Mutasyon testi çalışmalarında tercih edilen başka bir programdır. 7338 kod satırı ve 60 test vakası içermektedir.

JTopas: Deneylere alınan dördüncü programdır. Rasgele metin verilerini ayrıştırma sorunu için küçük, kullanımı kolay bir Java kitaplığı sağlar. Veriler, birkaç yorum, HTML, XML veya RTF Akışı, farklı programlama dillerinin kaynak kodu vb. içeren basit bir yapılandırma dosyasından gelebilir. 2775 kod satırı ve 74 test vakası vardır.

XStream: XStream, nesneleri XML'e seri hale getirmek ve tekrar geri yüklemek için basit bir kütüphanedir. Yaygın kullanım durumlarını basitleştiren yüksek düzeyde bir cephe temin edilmiştir. Çoğu nesne, eşleme belirtmeye gerek kalmadan serileştirilebilir. Hız ve düşük bellek alanı, tasarımın çok önemli bir parçasıdır ve büyük mesaj grafikleri veya mesaj verimi yüksek sistemler için uygundur.

Jaxen: Jaxen, Java ile yazılmış açık kaynaklı bir XPath kütüphanesidir. DOM, XOM, dom4j ve JDOM dahil olmak üzere birçok farklı nesne modeline uyarlanabilir. Derlenmiş Java bayt kodu veya Java fasulyesi gibi XML olmayan ağaçlara XML gibi davranan adaptörler yazmak da mümkündür, böylece bu ağaçları XPath ile sorgulamanızı da sağlar.

Commons-lang: Standart Java kütüphaneleri, çekirdek sınıflarının manipülasyonu için yeterli yöntemleri sağlamada başarısız olur. Apache Commons Lang, bu ekstra yöntemleri sağlar. Lang, java.lang API'si için özellikle Dize işleme yöntemleri, temel sayısal yöntemler, nesne yansıması, eşzamanlılık, oluşturma ve serileştirme ve Sistem özellikleri için birçok yardımcı yardımcı program sağlar. Ek olarak, java.util.Date'deki temel geliştirmeleri ve hashCode, toString ve eşittir gibi yapım yöntemlerinde yardımcı olacak bir dizi yardımcı program içerir.

3.3 Geçerliliğe Yönelik Tehditler

Bu kısımda mutasyon test probleminin üstesinden gelmek için önerilen yaklaşımın geçerliliğine yönelik tehditler sunulmaktadır.

3.3.1 İç geçerlilik

İç geçerliliğe yönelik tehditler, sonuçları etkileyebilecek veya bunlardan sorumlu olabilecek kontrolsüz faktörler olarak kabul edilir. Bu tezde, iç geçerliliğe yönelik tehdit, deneysel uygulamalarında hata yapma ve mutant üretme ihtimalinden kaynaklanmaktadır. Bu tehdidi azaltmak için, mutant üretiminde MuClipse kullanıldı. Buna ek olarak, herhangi bir deney yapılmadan önce tüm kodlar dikkatli bir şekilde gözden geçirilmektedir.

3.3.2 Dış geçerlilik

Dış geçerlilik tehditleri esas olarak konu programları ve mutasyon aracı ile ilgilidir. Bu tezde yapılan çalışmada, bu tehdidi azaltmak için, genellikle Java'da ve hatta C ++ 'da kullanılan çok çeşitli yapılar ve veriler içeren yedi konu program seçildi. Deneyde kullanılan programlar karşılaştırılabilir ve temsili programlardan seçilmiştir. Farklı alanlardan yedi gerçek dünya java programı seçilmesine rağmen, bu konular özellikle diğer programlama dillerinde başka projeleri temsil etmeyebilir. Ancak, bu çalışmadaki konu programlarında ele alınmayan yapılara sahip daha fazla konu kullanılarak daha fazla deneme yapılmasıyla bu tehdit daha da azaltılabilir. Mutasyon üretme aracı ayrıca mutantların kalitesini ve sayısını da etkiler. Deneysel çalışmalarda mutasyon aracı olarak, diğer mutasyon araçlarını temsil etmeyebilen MuClipse kullanıldı. Bu tehdidin daha da azaltılması için, mutant üretiminde başka mutasyon araçları da kullanılabilir.

3.3.3 Yapı geçerliliği

Yapı geçerliliğine yönelik temel tehdit, bu çalışmada eşdeğer mutantların tespit edilme şeklidir. Eşdeğer mutantları otomatik olarak belirlemek, onu tespit eden bir araç veya tüm eşdeğer olmayan mutantları tanımlayan bir test seti gerektirir. Bu tehdidi azaltmak için, mevcut eşdeğer mutant dedektör aracı olmadığından çalışmada eşdeğer olmayan mutantları tespit eden bir test yazıldı. Test vakası önceliklendirme aşamasında yapı geçerliliğine yönelik tehditi azaltmak için toplam blok kapsamı önceliklendirme tekniği

kullanılmıştır. Mutasyon kümelenme aşamasında ise K-means kümeleme algoritması seçilmiştir. Bu aşamadaki yapı geçerliliğine yönelik tehdit, uzaklık ölçütlerinin değerlendirilmesidir. Bu tehditi azaltmak için düzenleme mesafesi kullanılmıştır. Diğer bir tehdit, deneylerde kullanılan mutantların etkinliğini ölçmek olabilir. Bunu azaltmak için, Barbosa vd [E. F. Barbosa vd 2001]'nın geniş çapta kullanılan ölçütü kullanıldı. Üretilen mutantların sayımı, MuClipse mutasyon test aracında tamamen otomatik olduğu için hiçbir zaman sorun olmadı.

4. ARAŞTIRMA BULGULARI

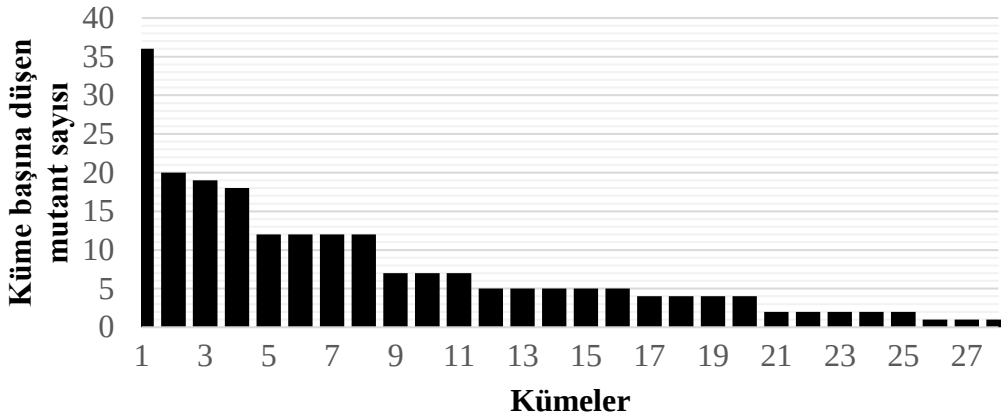
Bu bölümün amacı, çalışılan yöntemin etkinliğini bildirmektir. İlk olarak, eşdeğer mutantların tespit edilip elimine edildiğinden ve test vakası önceliği yürütme süresininin azaltığından emin olmak için, önceden tanımlanmış kritere göre değerlendirildi. Deneysel sonuçların gösterdiği gibi, bu aşamada eşdeğer mutantlar başarıyla tespit edilip elimine edildi. Daha sonra mutant sayısını azaltmak için mutasyon kümelenmesini kullanılarak yüksek mutasyon skoru elde edilmektedir.

Triangle Programı: Eşdeğer mutantların tespiti ve 18 test vakasının 216 mutanta karşı yürütüldüğü Üçgen programının test vakası önceliklendirilmesinin ardından, Çizelge 4.1'in gösterdiği gibi uygulama süresi neredeyse %50'ye düşürüldü. 216 mutanttan 120'si öldürülürken, 96 mutant, deneylerin test vakası önceliklendirme aşamasında hayatta kaldı. % 95'in çok altında optimal bir skoru olmayan 55 mutasyon skoru elde edildi. Bu program için eşdeğer bir mutant bulunamadığını not etmek gerekir.

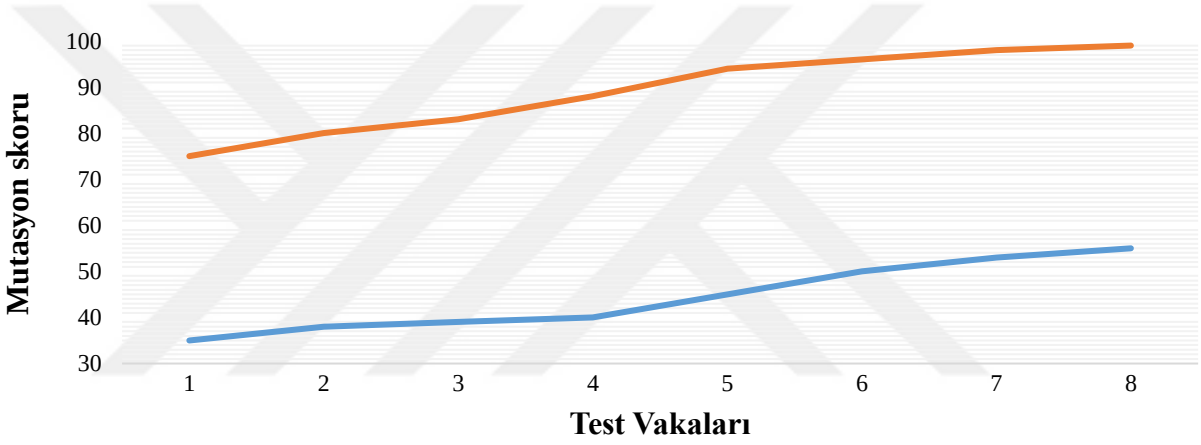
Çizelge 4.1 Eşdeğer mutant eliminasyonu ve Test vakası Önceliklendirmesi adımlarından sonra Üçgen programının sonucu

Programı	Toplam mutant sayısı	Toplam Tc sayısı	Öldürülen mutantlar	Yaklaşım öncesi çalısma süresi	Yaklaşım sonrası çalısma süresi	Mutasyon skoru
Triangle	216	18	120	32 saniye	15 saniye	55%

Daha fazla mutasyon azaltımı ve mutasyon skorunun arttırılması amacıyla, 216 mutantın üçgen program kümelemesi yapıldı. K-means algoritmaları kullanılarak mutantlar gruplandı ve kümelemenin ilk çalışmasında mutantlar 28 kümeye ayirildi. Daha sonra, ikinci ve üçüncü çalışmada kümeler sıraysıla 13 ve 7'ye düşürülmüştür.



Şekil 4.1 K-means algoritması kullanılarak Üçgen program mutantlarının kümeleri.



Şekil 4.2 Tüm aşamalar uygulanınca Üçgen programın sonucu, k=28

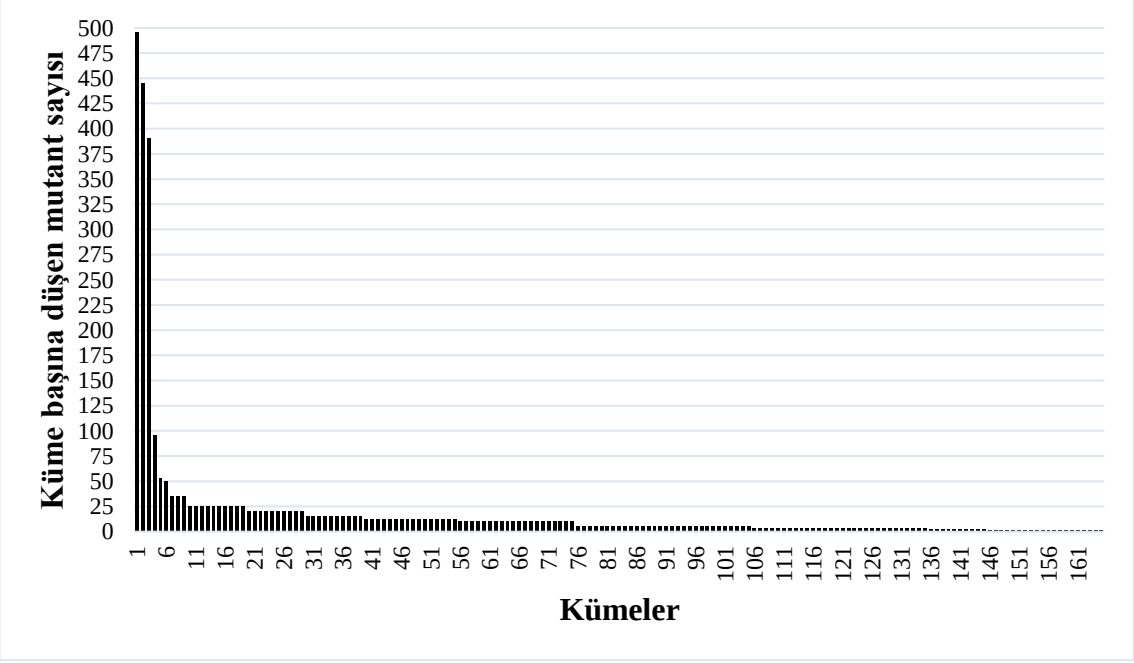
Mavi: Test vakası önceliğini gösterir, Kırmızı: Test vakası önceliği ve eşdeğer mutant tespiti ile mutant kümelenmesi. (not: bu programda eşdeğer mutant tespit edilmemiştir). Örnek olarak, yukarıdaki şekil 4.2'de Üçgen programında yürütülen denemenin iki örnek çalışma sonucu gösterilmektedir. İlk örnekte (mavi), eşdeğer mutantlarının tespiti ile elde edilen sonuç ve mutasyon skoru % 55 olan test vakası önceliği gösterilmiştir. İkinci örnekte (kırmızı), eşdeğer mutant tespitinden sonra deney programında test vakası önceliklendirme ve mutasyon kümelenmesi uygulanarak ulaşılan mutasyon skorunun sonucu sunulmaktadır. Burada mutasyon skoru % 99.4 yüksektir.

Time & Money Programı: Eşdeğer mutantların tespit aşamasında mutantlar 2320'den 2100'e azaltıldı, bu da 220 eşdeğer mutantların elimine edildiği anlamına gelmektedir. Elimine edilen mutantlar, herhangi bir test vakası tarafından erişilmeyen eşdeğer mutantlardır. Ardından test vakası önceliklendirme aşamasında mutantlar 1300'e düşürüldü. Test vakaları da 227'den 190'a azaltıldı. Uygulama süresi 3 dakikadan 45 saniye'ye düşürüldü.

Çizelge 4.2 Eşdeğer mutant eliminasyonu ve Test vakası Önceliklendirmesi adımlarından sonra Time & Money Programın sonucu.

Programı	Toplam mutant sayısı	Toplam Tc sayısı	Öldürülen mutantlar	Elimine edilmiş mutantlar	Yaklaşım öncesi çalısma süresi	Yaklaşım sonrası çalısma süresi	Mutasyon skoru
Time & Money	2320	227	1300	220	3 dakika	45 saniye	43%

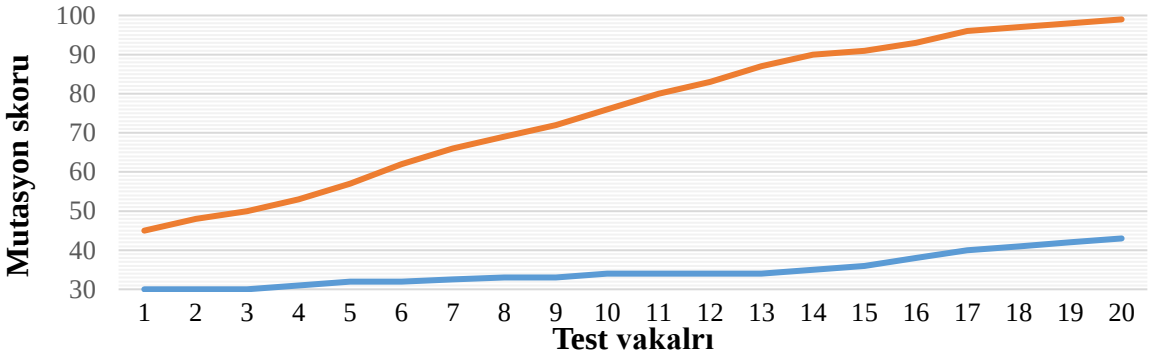
Daha fazla mutant azaltma amaçıyla ve yüksek mutasyon skoru performansına ulaşmak için, Time & Money programının mutantlarına, K-means kümeleme algoritması kullanılarak kümeleme uygulanmıştır. Şekil 4.3'te görüldüğü gibi ilk çalışmada mutantlar 165 kümeye kümelenmiştir. Daha sonra kümelenmeler üzerinde 75, 40 ve 15'lik bir azalma elde edildi.



Şekil 4.3 K-means algoritması kullanarak Time/Money program mutantlarının kümeleri

Şekil 4.4, eşdeğer tespiti ve test vakası önceliklendirmesinden sonra, 20 test vakası ve 165 kümeye sahip olan Time & Money programına uygulanan bir kümelenme yöntemi

çalışmasını göstermektedir. Ayrıca, eşdeğer mutant tespiti ile test vakası önceliklendirme uygulanıp bir kümelenme yöntemi uygulanmayan başka bir örnek çalışmasını da sunmaktadır. Test vakasının önceliklendirilmesi ile eşdeğer tespiti mutasyon skoru, yukarıdaki şekilde gösterildiği gibi % 43 iken, tüm fazlar uygulanınca % 99'luk yüksek bir mutasyon skoru elde edilmiştir. Söylemeye değer bir başka şey ise test durumunun 227'den 20'ye düşürüldüğüdür.

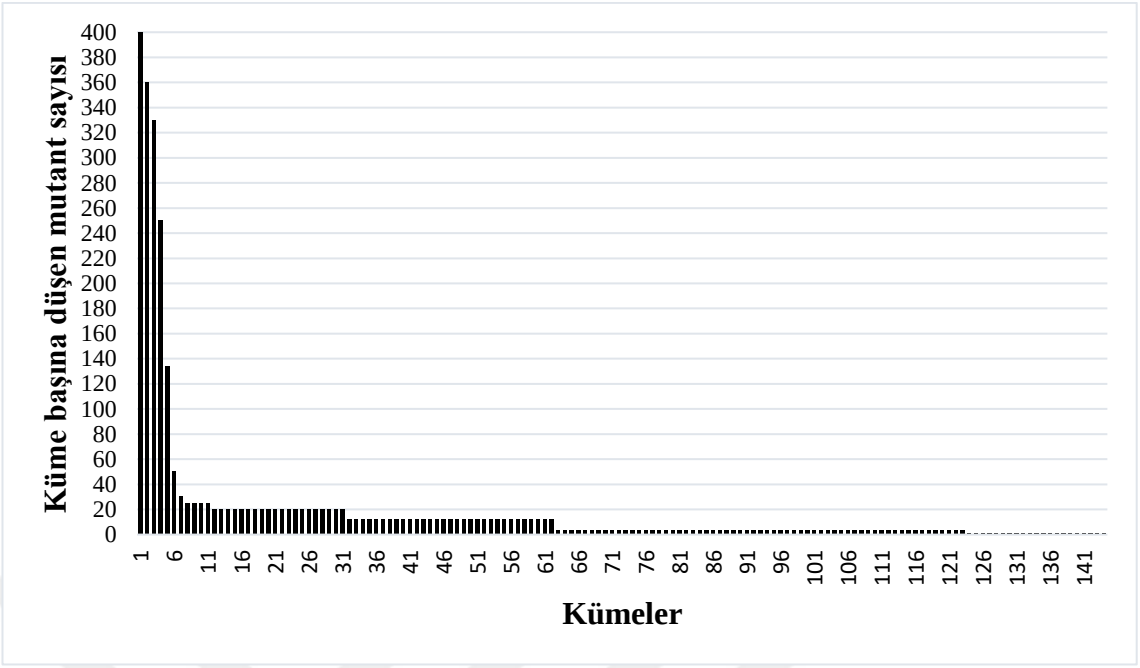


Şekil 4.4 Tüm aşamalar uygulanınca time/money programın sonucu, k=165.
Mavi: Test vakası önceliğini gösterir, Kırmızı: Test vakası önceliği ve eşdeğer mutant tespiti ile mutant kümelenmesi.

JDepend Programı: Tespit aşamasında eşdeğer olarak 250 mutant saptandı. Dahası, uygulama süresini en aza indirmek için bu programa test vakası önceliklendirilme uygulandı. Bu aşamada mutantlar 2475'den 1505'e düşürüldü, bu da mutantların neredeyse yarısının elimine edildiği ve öldürüldüğü anlamına gelmektedir. Uygulama süresi de 3.5 dakikadan 53 saniyeye azaltıldı.

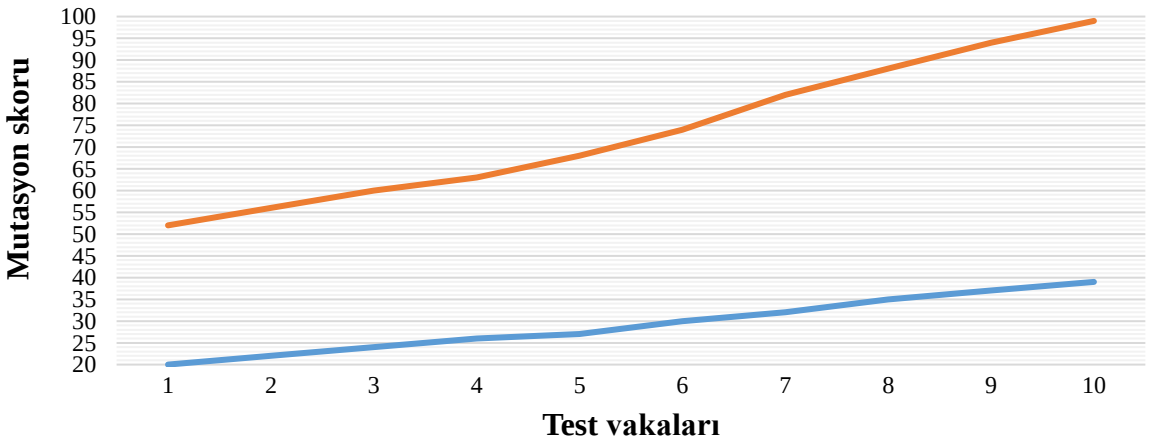
Çizelge 4.3 Eşdeğer mutant eliminasyonu ve test vakası önceliklendirmesi adımlarından sonra JDepend Programın sonucu

Programı	Toplam mutant sayısı	Toplam Tc sayısı	Öldürülen mutantlar	Elimine edilmiş mutantlar	Yaklaşım öncesi çalısma süresi	Yaklaşım sonrası çalısma süresi	Mutasyon skoru
JDenend	2725	60	970	250	3.5 dakika	53 saniye	39%



Şekil 4.5 K-means algoritması kullanarak JDepend program mutantlarının kümeleri

Eşdeğer mutantlar tespit edildikten ve test vakası önceliklendirilmesinden sonra JDepend programına K-means kümeleme yöntemi uygulandı. Bu aşamada mutantlar, Şekil 4.5'in gösterdiği gibi ilk çalışmada 143 kümeye kümelenmiştir. Daha sonra kümeler 65, 30 ve 15'e düşürüldü.



Şekil 4.6 Tüm aşamalar uygulanınca JDepend programın sonucu, k=143.

Mavi: Test durumu önceliğini gösterir, Kırmızı: Test durumu önceliği ve eşdeğer mutant tespiti ile mutant kümelenmesi.

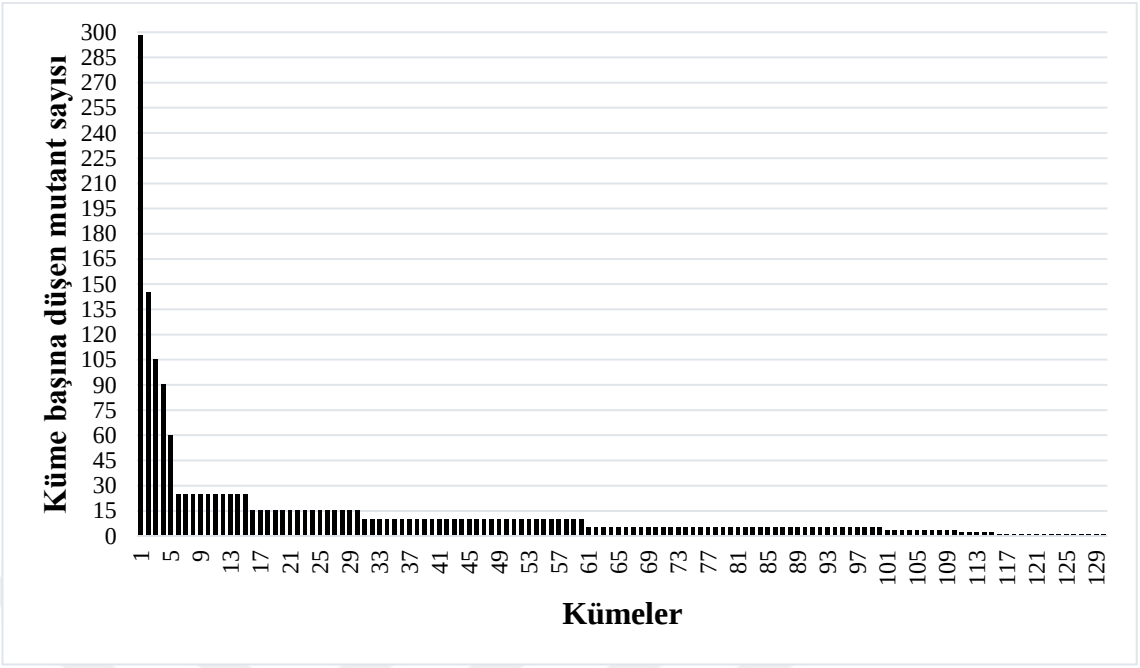
Şekil 4.6'da, eşdeğer mutantları tespit edilip test vakasına öncelik verildikten sonra kümeleme algoritmalarına uygulanan bir JDepend programı çalışması gösterilmektedir. 20 test vakası ile 143 küme kullanarak ,% 99'luk optimal bir mutasyon skoruna ulaşıldı. Test vakaları ayrıca 10'a düşürülmüştür. Şekil 4.6'da aynı zamanda kümeleme algoritması uygulanmadan sadece test vakası önceliklendirme ile elde edilen çok düşük bir mutasyon skoruna sahip olan başka bir çalışma da gösterilmiştir.

JTopas Programı: Dördüncü programda 1753 mutant ve 74 test vakası bulunmaktadır. Herhangi bir test vakasıyla ulaşılamayan mutantları elimine etmek için JTopas programına eşdeğer mutant tespiti uygulandı ve ardından yürütme süresini en aza indirmek için test vakasına öncelik verildi. Sonuç olarak, yaklaşımın ilk aşamasında 55 eşdeğer mutant elimine edildi. Test vakası önceliklendireme aşamasında 1060 mutant öldürüldü ve uygulama süresi 35 saniyeye indirildi.

Çizelge 4.4 Eşdeğer mutant eliminasyonu ve test vakası önceliklendirmesi adımlarından sonra Jtopas Programın sonucu

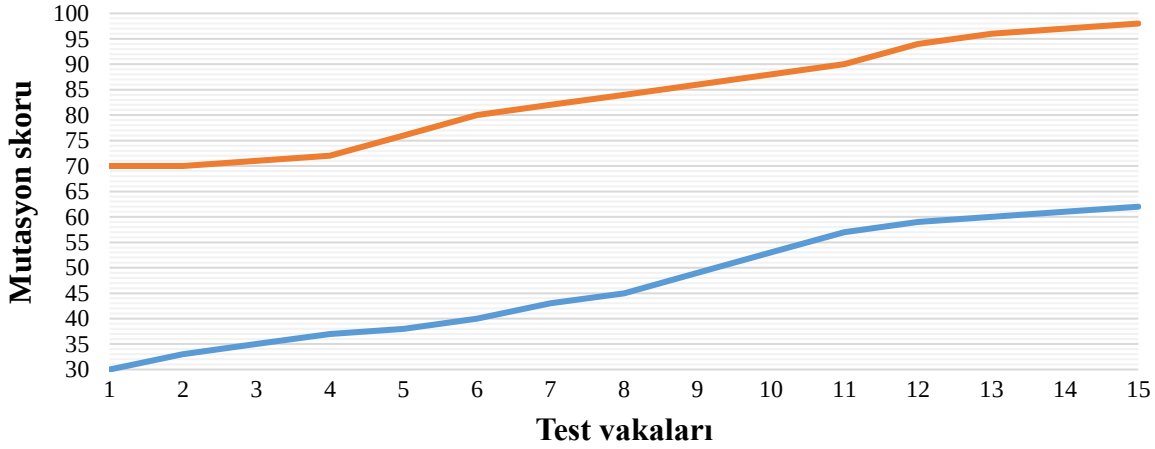
Programı	Toplam mutant sayısı	Toplam Tc sayısı	Öldürülen mutantlar	Elimine edilmiş mutantlar	Yaklaşım öncesi çalısma süresi	Yaklaşım sonrası çalısma süresi	Mutasyon skoru
JTopas	1753	74	1060	55	2.15 dakika	35 saniye	62%

Şekil 4.7’de kümeleme aşamasında JTopas programının mutantları aralarındaki mesafeye göre 135 kümeye ayırıldığı gösterilmektedir. Daha sonra kümeler 65, 35 ve 15 kümeye düşürüldü.



Şekil 4.7 K-means algoritması kullanarak Jtopas program mutantlarının kümeleri

Şekil 4.8'de, kümeleme algoritması ile eşdeğer mutantları tespit etme ve test vakası önceliklendirmesi uygulanan Jtopas programının iki örnek çalışması gösterilmektedir. Birinci çalışmada önerilen yaklışımın tüm aşamaları kullanılırkan, ikinci çalışma da ise sadece eşdeğer mutantları tespit etme ve test vakası önceliklendirmesi uygulandı. Her iki çalışma eşit sayıdaki 15 test vakasına sahiptir. Bu programda tüm fazlar uygulanınca %98'luk yüksek bir mutasyon skoru elde edilirken sadece eşdeğer mutantları tespit etme ve test vakası önceliklendirmesi uygulandığında düşük bir mutasyon skoru elde edildi.



Şekil 4.8 Tüm aşamalar uygulanınca Jtopas programın sonucu, k=135.

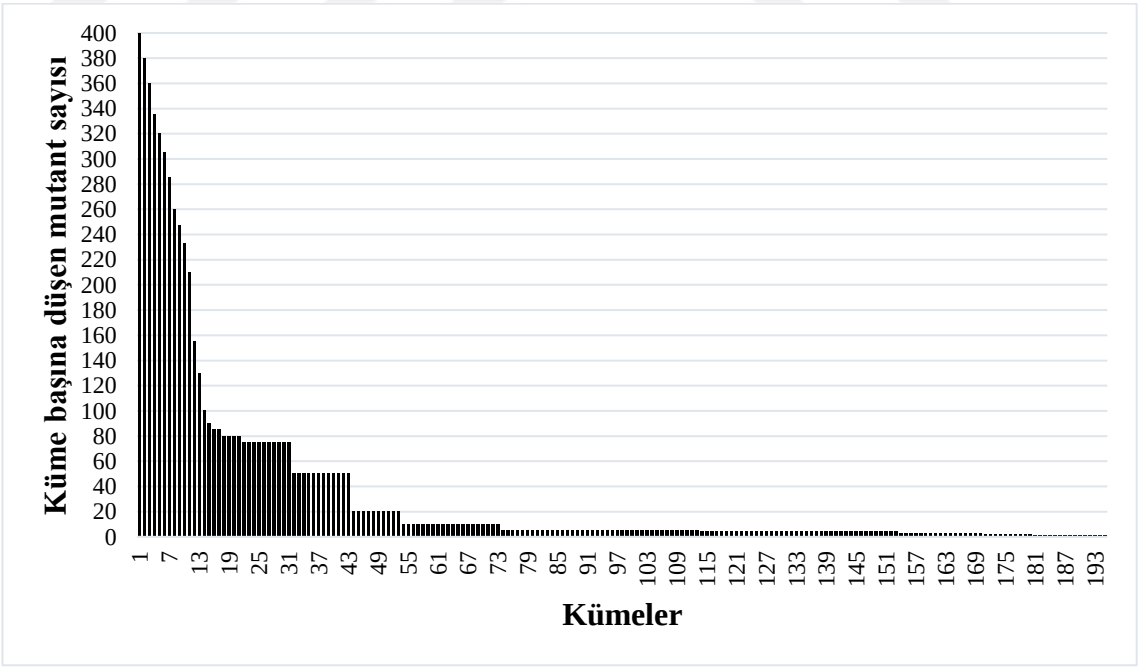
Mavi: Test durumu önceliğini gösterir, Kırmızı: Test durumu önceliği ve eşdeğer mutant tespiti ile mutant kümelenmesi.

XStream Programı: eşdeğer tespit aşamasında 600 mutant elimine edilmiştir. Ardından test vakası önceliklendirmede mutantlar 7,450'dan 2,200'ye düşürülmüştür. Bu iki aşamada %70 düşük bir mutasyon skoru elde edilmiştir. önemli noktalardan bir tanesi de, test süresi 9.15 dakikadan 4.35 dakika'ye düşürülmüştür.

Çizelge 4.5 Eşdeğer mutant eliminasyonu ve test vakası önceliklendirmesi adımlarından sonra XStream programın sonucu

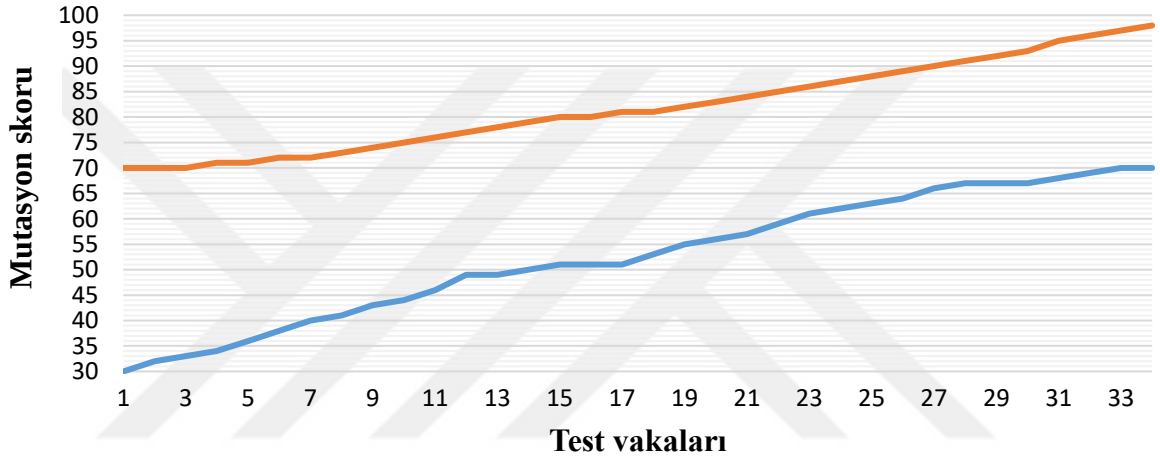
Programı	Toplam mutant sayısı	Toplam Tc sayısı	Öldürülen mutantlar	Elimine edilmiş mutantlar	Yaklaşım öncesi çalısma süresi	Yaklaşım sonrası çalısma süresi	Mutasyon skoru
XStream	8,050	340	5250	600	9.15 dakika	4.35 dakika	70%

Şekil 4.9, kümeleme aşamasında XStream programının mutantlarını aralarındaki mesafeye göre 195 kümeye ayırdığını göstermektedir. Daha sonra kümeler 87, 43 ve 21 kümeye düşürüldü.



Şekil 4.9 K-means algoritması kullanarak XStream program mutantlarının kümeleri

Şekil 4.10'da, kümeleme algoritması ile eşdeğer mutantları tespit etme ve test vakası önceliklendirmesi uygulanan XStream programının iki örnek çalışması gösterilmektedir. Birinci çalışmada önerilen yaklışımın tüm aşamaları kullanılırkan, ikinci çalışma da ise sadece eşdeğer mutantları tespit etme ve test vakası önceliklendirmesi uygulandı. Her iki çalışma da eşit sayıdaki 33 test vakasına sahiptir. Tüm aşamalar uygulanıca %98'luk yüksek bir mutasyon skoru elde edilmiştir. Eşdeğer mutantları tespit etme ve test vakası önceliklendirme aşamalarında ise %70 düşük bir mutasyon skoru elde edildi.



Şekil 4.10 Tüm aşamalar uygulanınca XStream programın sonucu, k=195.

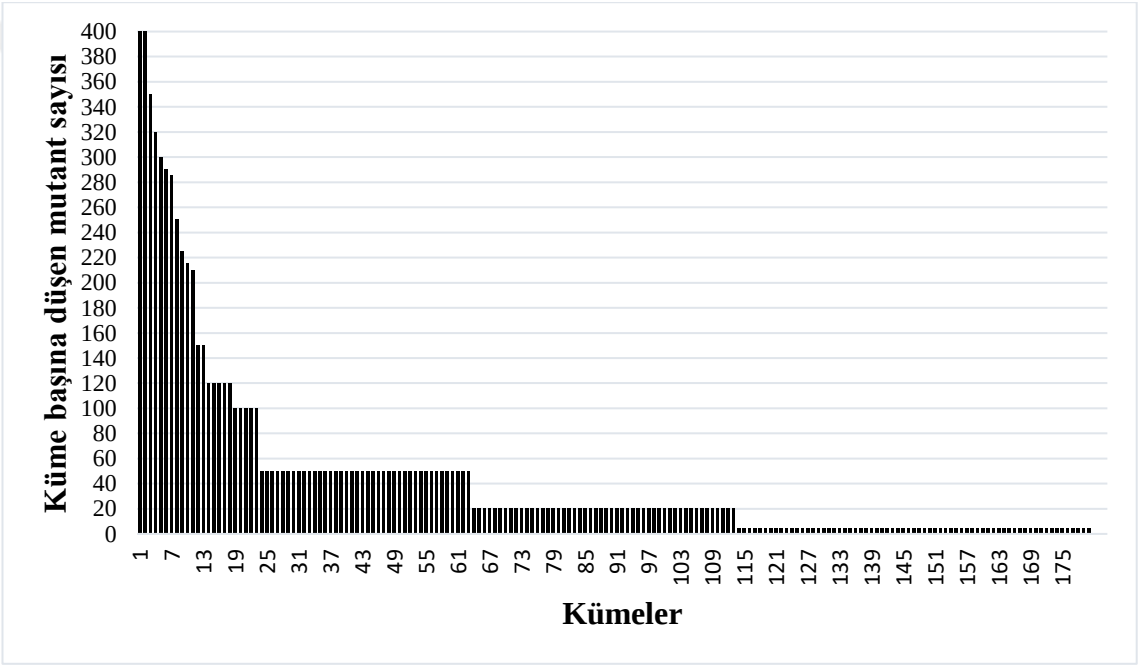
Mavi: Test vakası önceliğini gösterir, Kırmızı: Test durumu önceliği ve eşdeğer mutant tespiti ile mutant kümelenmesi.

Jaxen Programı: Eşdeğer mutantların tespit aşamasında mutantlar 7125'den 6795'e azaltıldı, bu da 330 eşdeğer mutantların tespit edildiği anlamına gelmektedir. Elimine edilen mutantlar, herhangi bir test vakası tarafından erişilmeyen eşdeğer mutantlardır. Ardından test vakası önceliklendirme aşamasında mutantlar 2345'e düşürülmüştür. Test süresi 8.45 dakikadan 3.50 dakika'ya düşürülmüştür.

Çizelge 4.6 Eşdeğer mutant eliminasyonu ve test vakası önceliklendirmesi adımlarından sonra Jaxen programın sonucu

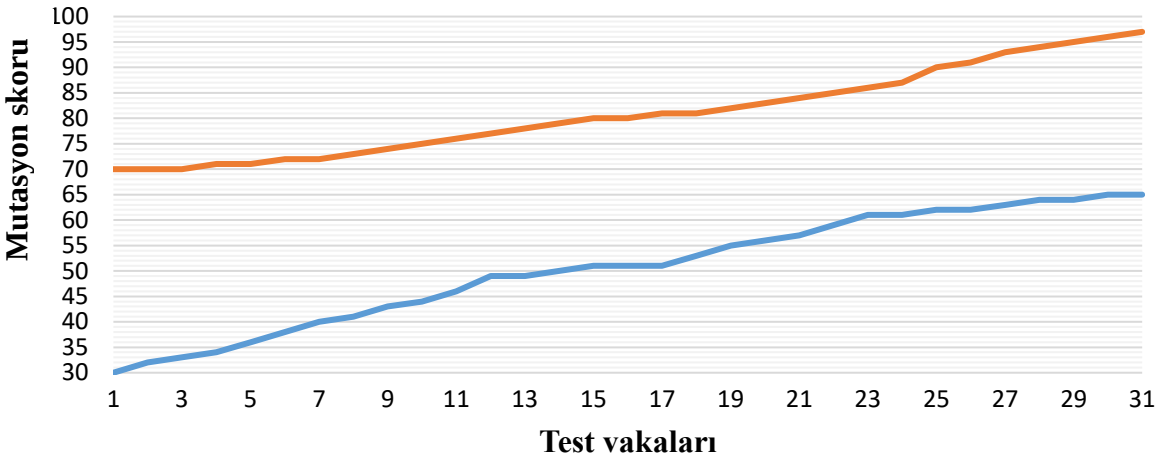
Programı	Toplam mutant sayısı	Toplam Tc sayısı	Öldürülen mutantlar	Elimine edilmiş mutantlar	Yaklaşım öncesi çalısma süresi	Yaklaşım sonrası çalısma süresi	Mutasyon skoru
Jaxen	7125	293	4450	330	8.45 dakika	3.50 dakika	65%

Şekil 4.11'de kümeleme aşamasında Jaxen programının mutantları aralarındaki mesafeye göre 175 kümeye ayırdığı gösterilmektedir. Daha sonra kümeler 80, 35 ve 15 kümeye düşürüldü.



Şekil 4.11 K-means algoritması kullanarak Jaxen program mutantlarının kümeleri

Şekil 4.12'de, kümeleme algoritması ile eşdeğer mutantları tespit etme ve test vakası önceliklendirmesi uygulanan Jaxen programının iki örnek çalışması gösterilmektedir. Birinci çalışmada önerilen yaklışımın tüm aşamaları kullanılırkan, ikinci çalışma da ise sadece eşdeğer mutantları tespit etme ve test vakası önceliklendirmesi uygulanmıştır. Deneyin ilk iki aşamasında %65 düşük bir mutasyon skoru elde edilmiştir. Tüm aşamalar uygulandığında ise %97.5'luk yüksek bir mutasyon skoru elde edilmiştir.



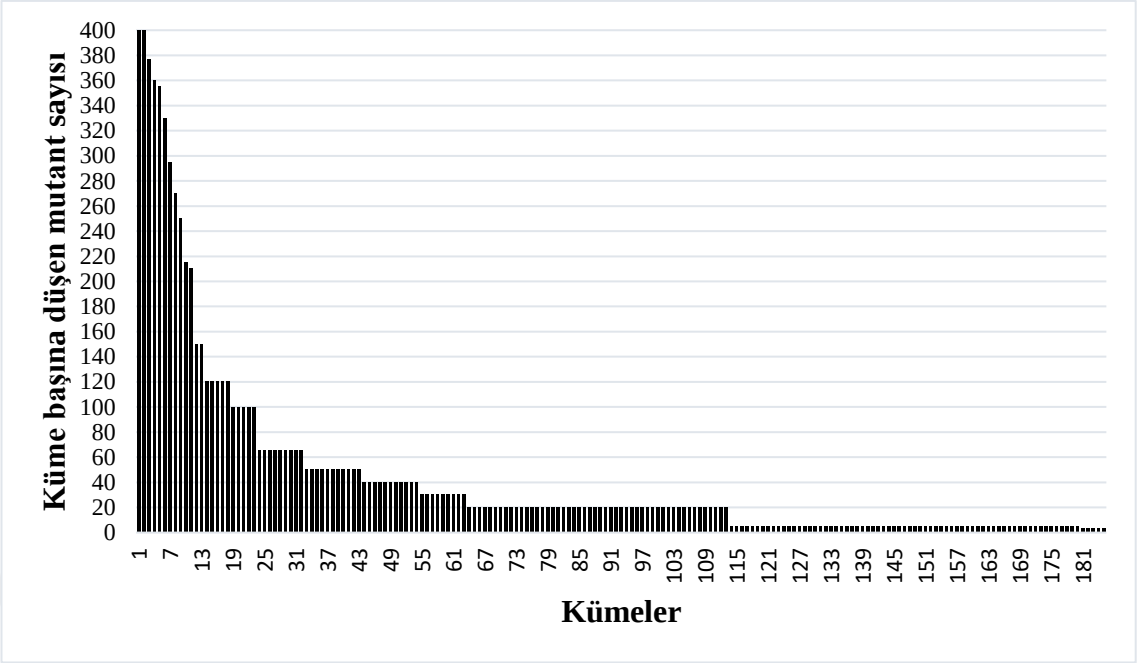
Şekil 4.12 Tüm aşamalar uygulanınca Jaxen programın sonucu, k=175.

Mavi: Test durumu önceliğini gösterir, Kırmızı: Test durumu önceliği ve eşdeğer mutant tespiti ile mutant kümelenmesi.

Commons-lang Programı: Deneyin son programında, eşdeğer mutantların tespit aşamasında mutantlar 7,520'den 7,065'e azaltıldı, bu da 455 eşdeğer mutantların tespit edildiği anlamına gelmektedir. Elimine edilen mutantlar, herhangi bir test vakası tarafından erişilmeyen eşdeğer mutantlardır. Ardından test vakası önceliklendirme aşamasında mutantlar 1,775'e düşürülmüştür. Test süresi 8.55 dakikadan 4 dakika'ya düşürülmüştür.

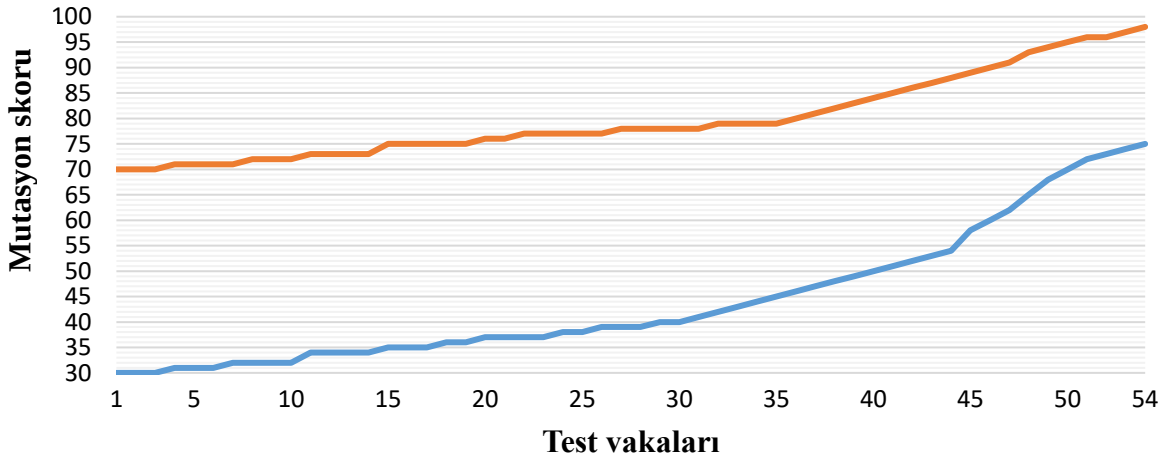
Çizelge 4.7 Eşdeğer mutant eliminasyonu ve test vakası önceliklendirmesi adımlarından sonra Commons-lang programın sonucu

Programı	Toplam mutant sayısı	Toplam Tc sayısı	Öldürülen mutantlar	Elimine edilmiş mutantlar	Yaklaşım öncesi çalısma süresi	Yaklaşım sonrası çalısma süresi	Mutasyon skoru
Commons-lang	7,520	375	5,290	455	8.55 dakika	4 dakika	75%



Şekil 4.13 K-means algoritması kullanarak Commons-lang program mutantlarının kümeleri

Şekil 4.13, kümeleme algoritmasının Commons-lang programının mutantlarını aralarındaki mesafeye göre 185 kümeye ayırdığını göstermektedir. Daha sonra kümeler 90, 40 ve 20 kümeye düşürüldü.



Şekil 4.14 Tüm aşamalar uygulanınca Commons-lang programın sonucu, k=185

Mavi: Test durumu önceliğini gösterir, Kırmızı: Test durumu önceliği ve eşdeğer mutant tespiti ile mutant kümelenmesi.

Şekil 4.14'te, kümeleme algoritması ile eşdeğer mutantları tespit etme ve test vakası önceliklendirmesi uygulanan son deney programının iki örnek çalışması gösterilmektedir. Birinci çalışmada önerilen yaklışımın tüm aşamaları kullanılırkan, ikinci çalışma da ise sadece eşdeğer mutantları tespit etme ve test vakası önceliklendirmesi uygulanmıştır. Tüm aşamalar uygulanınca %98'luk yüksek bir mutasyon skoru elde edilirken önerilen yaklaşımın ilk iki aşamasında %75 etkili olmayan düşük bir mutasyon skoru elde edildiği görülmektedir.

5. TARTIŞMA VE SONUÇ

Bu bölümün ilk kısmında araştırma bulguları hakkında değerlendirme yapılmaktadır. Daha sonra önerilen yaklaşım uygulanarak elde edilen sonuç ve öneriler vurgulanmaktadır.

5.1 Değerlendirme

Bu kısımda deneysel çalışma bulguları değerlendirip çalışma alanında daha önce yapılan araştırma bulguları ile karşılaştırmaktadır.

İlk olarak, eşdeğer mutantları tespit etmek ve sonra bunları elimine etmek. Aslında, tüm eşdeğer mutantları tespit etmek zordur ve bir şekilde imkansızdır. Bununla birlikte, bu çalışmada yedi konu programın altı tanesinde eşdeğer mutantlar tespit edilip elimine edilmektedir. Çizelge 4.1'de gösterildiği gibi, 216 mutant ve 18 test vakasına sahip olan Üçgen programında eşdeğer mutantlar tespit edilmemiştir. Büyük programlar söz konusu olduğunda, önerilen yaklaşımın eşdeğer mutantların tespitinde iyi çalıştığı anlaşılmaktadır. Time ve Money programındaki 2320 mutantın 220'sinin eşdeğer mutantlar olarak tespit edilip ortadan kaldırıldığı ğibi, Çizelge 4.1 bu iddiayı desteklenmektedir. Çizelge 5.1'de gösterildiği gibi JDepend programında 250, JTopas programında ısa 55 eşdeğer mutant testpit edilip elimine edilmiştir.

İkincisi, yürütme süresini en aza indirmek. Mutasyon testinde, bir test vakasının her bir mutanta karşı çalıştırılması gerekir ve bu, uygulama süresini arttırır. Önerilen yaklaşım, mutantların çoğunu öldüren test vakalarına ilk çalışma önceliği vererek programın büyüklüğüne bakmaksızın uygulama süresini etkili bir şekilde azaltmaktadır. Deneylerin ilk konu programında, test vakalarına öncelik verilmeden önceki yürütme süresinin neredeyse % 50'si azaltılarak 15 saniyeye düşürülmüstür. Time & Money programında yürütme süresi 3 dakikadan 45 saniyeye düşürülürken, JDepend programında 3,5 dakikadan 53 saniyeye indirilirmiştir. Test vakası önceliklendirme aşamasında bazı konu programlarının test süresi yarıya düşerken test vakaları ise çok düşüyor. Bunun nedeni bazı test vakalarının birden fazla mutant öldürmesi ve bir mutantın birden fazla test vakası tarafından öldürülmesidir. Mesela Çizelge 3.1'de bir programının altı tane mutantı ile altı tane test vakası vardır. Bu programa test vakası önceliklendirme uygulandıktan sonra test vakası %33 düşerken test süresi ise o kadar

düşmemektedir. Aynı durum deneyde kullınılan bazı programlar için geçerlidir mesala time and money.

Üçüncüsü, herhangi bir mutasyon operatöründen kaçınılmadan veya çıkarılmadan mutantların sayısını azaltmak. Çok sayıda mutantın bulunması, en büyük olmasa da mutasyon testinin karşılaştığı en büyük zorluklardan biridir. Önerilen yaklaşım, mutantları öldürebilen test vakalarına dayanarak, onları farklı gruplara ayırıp mutantların sayısını azaltmakta etkilidir. Üçgen programına bakılırsa, toplam 216 mutantı vardır, ilk çalıştırmada 28'e düşürülmüştür. Daha sonra,% 99'luk yüksek mutasyon skoruna ulaşmayı mümkün kılan 13 ve 7'ye indirilmiştir. Ek olarak, test vakaları da anlamlı bir test etkinliği kaybı olmadan 18'den 8'e düşürüldü. Şaşırtıcı bir şekilde, deneylerde, programlarının boyutuna bakılmaksızın mutantların sayısını azaltmakta yaklaşımın etkili olduğu gözlemlenmektedir. Şekil 4.3, eşdeğer mutantları elimine ettikten sonra 2100 mutant içeren Time & Money programının kümeleme işleminin bir örneğini göstermektedir. Mutantlar 165'e kümelenmiştir ve daha sonra 75, 40 ve 15'e düşürülmüştür. Test vakaları da 227'den 20'ye indirilip % 99'luk mutasyon skoru elde edilmiştir.

Çizelge 5.1 Test vakasi önceliği ve eşdeğer mutant tespiti ile yapılan yaklaşımın özet sonucu.

Programı	Toplam mutant sayısı	Toplam Tc sayısı	Öldürülen mutantlar	Elimine edilmiş mutantlar	Yaklaşım öncesi çalısma süresi	Yaklaşım sonrası çalısma süresi	Mutasyon skoru
Triangle	216	18	120	0	32 saniye	15 saniye	55%
Time & Money	2320	227	1300	220	3 dakika	45 saniye	43%
JDenend	2725	60	970	250	3.5 dakika	53 saniye	39%
JTopas	1753	74	1060	55	2.15 dakika	35 saniye	62%
XStream	8,050	340	5250	600	9.15 dakika	4.35 dakika	70%
Jaxen	7125	293	4450	330	8.45 dakika	3.50 dakika	65%
Commons-lang	7,520	375	5,290	455	8.55 dakika	4 dakika	75%

Çizelge 5.1 eşdeğer mutant tespiti ve test vakası önceliği sonrası yaklaşımın özet sonucunu göstermektedir. Eşdeğer mutant tespiti aşamasında, Üçgen programı (Triangle program) hariç tüm konu programlarında eşdeğer mutantlar tespit edilip elimine edilmiştir. Bu çizelgedeki mutasyon skorunun çizelge 3.2'ye göre arttığı görülmektedir. Bunun nedeni, saf mutasyon testinin çizelge 3.2'deki deneysel çalışmada uygulanmasıdır. Test vakası önceliklendirme aşamasında ise, tüm konu programları için test çalıştırma süresi neredeyse yarısına kadar düşürülmüştür.

Çizelge 5.2 Bütün fazlar uygulandığında elde edilen sonucun özeti.

Program	Toplam mutant sayısı	Toplam Tc sayısı	Indirgenmiş Tc sayısı	Yaklaşım öncesi çalısma süresi	Çalısma süresi	Mutasyon skoru
Triangle	216	18	8	32 saniye	19 saniye	99.4%
Time & Money	2,320	227	20	3 dakika	1.5 dakika	99%
JDepend	2,725	60	10	3.5 dakika	2 dakika	99%
JTopas	1,753	74	15	2.15 dakika	1.10 dakika	98%
XStream	7,450	340	34	9.15 dakika	2.15 dakika	98%
Jaxen	6,795	293	31	8.45 dakika	2.05 dakika	97.4%
Commons-lang	7,065	375	54	8.55 dakika	1.57 dakika	98%

Çizelge 5.2'de bütün fazların uygulanması yaklaşıymla elde edilen sonuc özet sunulmaktadır. Yaklaşımın üçüncü aşamasında, mutantların sayısını daha da azaltmak için mutasyon kümelenmesi uygulanmaktadır. Mutantlar, öldürülebilir test vakalarına göre kümelenmektedir ve her kümedeki mutantların aynı test vakaları tarafından öldürülme garantisi vardır. Her kümeden çok az sayıda mutant seçilmektedir ve böylece yüksek mutasyon skorunu elde etmek için mutantlar ve test vakaları indirgenmektedir. Mutasyon kümelenmesinin dezavantajları başlangıç değerlerine bağımlı olmak, aykırı olanları tespit etmemek ve merkez kavramına sahip verilerle sınırlı olmaktır.

Çizelge 5.3 Daha önce yapılan çalışmlarla elde edilen sonuç özeti

Programı	Çalışma	Mutasyon skoru
Triangle	S. Hussain. 2008	%98.9
Time & Money	Zhang, Lingming vd. 2013	%99
JDepend	Zhang, Lingming vd. 2013	%98
JTopas	Zhang, Lingming vd. 2013	%99
XStream	David Schuler 2011	%71
Jaxen	David Schuler 2011	%48
Commons-lang	David Schuler 2011	%80

Önerilen yaklaşım uygulanarak elde edilen sonuçlar, daha önce yapılan başka çalışmaların bulgularıyla karşılaştırılığında, seçici mutasyonun ilk çalışmalarından biri olan Offutt vd. [A.J. Offutt. 1993] çalışmasında 10 küçük Fortran-77 programında deneyler yapılmıştır. Bu programlar, aslında seçici mutasyonun ölçeklenebilirliğine bir soru işareti koyan 10 ila 48 çalıştırılabilir ifadelerdir. Namin vd. seçici mutasyonla ilgili başka bir çalışma olan [A.S. Namin 2008], 11 Siemens programı değerlendirilmiştir. Bu programlar 137 - 513 kod satırını kapsar, bu da sonuçları yeterli kılmayacak küçük programlar oldukları anlamına gelir. Barbosa vd. [E. F. Barbosa 2001], deneylerini toplam 619 cümleye sahip 27 C programı ile gerçekleştirdiler. Küçük programları kullanma ve değerlendirme eğiliminde oldukları görülmektedir. Yu-Seung ve Sang-Woon [Yu-Seung ve Sang-Woon 2018] çalışmalarında ise, mutantların kümelenmesini herhangi bir kümeleme algoritması kullanmadan gerçekleştirdi ve eşdeğer mutant göz ardı edilmiştir.

Sonuçta, daha önce yapılan çalışmalar ile bu çalışmanın arasındaki temel fark şudur: İlk önce, bu araştırmada çok çeşitli kontrol ve veri yapısına sahip daha geniş konu programları kullanılmaktadır. Küçük programlar üzerinde deneyler yapmak, mutasyon testlerinde küçük bir programdan daha büyük bir program çalıştırmanın daha maliyetli olabileceğini işaret ediyor. İkincisi, aynı anda mutantların sayısının azaltılması ve eşdeğerlerinin tespit edilmesi ile ilgili türünün ilk çalışmasıdır. Son olarak önemli noktalardan bir tanesi de, önceki tüm çalışmalarda yapılandırılmış dillere odaklanıldı, bu çalışmada ise Java dilinde kümeleme algoritmasını kullanılarak mutasyon maliyetinin düşürülmesine odaklanıldı.

5.2 Sonuç

Bu araştırma, mutasyon testinin bazı temel sorunlarının (çok sayıda mutant, çok fazla eşdeğer mutant oluşumu ve uygulama süresi) üstesinden gelmek için mutasyon maliyetini düşürme yaklaşımı sunar. Deneysel sonuçlar, önerilen yaklaşımın mutant kümesinin kuvvetini düşürmeden, çok sayıda mutantı azaltmada etkili olduğunu göstermektedir. Deneylerin ilk aşamasında -eşdeğer mutantların tespit edilmesi- herhangi bir test vakasıyla ulaşılamayan mutantlar, mutant kümesinden başarıyla elimine edildi. Polo vd. tarafından önerilen algoritmalardan birini kullanarak, [Polo vd 2009], Üçgen programı dışında tüm deney konu programlarında eşdeğer mutantlar tespit

edilip çıkartıldı. Deneyin eşdeğer mutantlarının tespit edilmesi sürecinde, eşdeğer mutantların tanımlanmasının LOC'daki artışla birlikte yükseldiği görülmüştür. Her bir mutanta karşı bir test vakası çalıştırmak, uygulama süresini arttırır, bunu azaltmak için, ilk olarak mutantların çoğunu kaplayan/öldüren test vakalarına ilk çalıştırma önceliği verilmiştir. Test vakasının önceliklendirilmesinin arkasındaki ana fikir, bazı kriterlere göre, önceden çalıştırılmak üzere yüksek önceliğe sahip test vakaları planlayarak test setinin erken arıza tespit oranını artırmaktır. Deneylerde, bu yöntem tüm konu programlarında yürütme süresinin neredeyse yarısını minimize ederek azaltmıştır.

Test vakası önceliklendirme yapıldıktan sonra, yüksek mutasyon skoru elde etmek için, daha fazla mutant ve test vakasının azaltılması amacıyla mutasyon kümelenmesi yapıldı. Kümeleme yönteminde mutantların sayısı en aza indirildi, ancak bununla birlikte testin boyutu da düşürüldü. Test kümesi sayısının azaltılmasına rağmen, yine de tüm mutantları öldürmek ve yüksek bir mutasyon skoru elde etmek için yeterince güçlü olduğunu belirtmekte fayda var. Mutantlar ve test vakaları bakımından farklı uzunluklarda ve değişkenlik gösteren yedi farklı java programında deneyler yapılmıştır. Tüm programlarda% 99'luk bir mutasyon skoru elde edildi.

Gelecekteki çalışmalar ve iyileştirmelerde, mutant kümesindeki eşdeğer mutantları tespit etmek, tanımlamak ve elimine etmek amacıyla kümeleme yöntemi genişletilip, geliştirilebilir. Örneğin bu tezde yapılan deney çalışmalarında ele alınmayan başka algoritmalar kullanılarak önerilen yaklaşım geliştirilebilir. Bu, önerilen yaklaşımı farklı aşamalardan oluşturmak yerine, yalnızca bir aşamaya sahip olmasına neden olabilir ve böylece deney süresi kısaltılabilir.

KAYNAKLAR

Anonymous. 2019a. Web Sitesi: http://timeandmoney.sourceforge.net/, Erişim Tarihi: 07.05.2019.

Anonymous. 2019b. Web Sitesi: http://jtopas.sourceforge.net/jtopas/index.html/, Erişim Tarihi: 07.05.2019.

Anonymous. 2019c. Web Sitesi: https://sourceforge.net/projects/jdepends/, Erişim Tarihi: 07.05.2019.

Anonymous. 2019d Web Sitesi: http://azhar-paperpresentation.blogspot.com/2010/04/software-testing-methodologies.html Erişim Tarihi: 07.05.2019.

Anonymous. 2019e Web Sitesi: https://x-stream.github.io/tutorial.html Erişim Tarihi: 07.05.2019.

Anonymous. 2019f Web Sitesi: https://sourceforge.net/projects/jaxen/ Erişim Tarihi: 07.05.2019.

Anonymous. 2019g Web Sitesi: https://commons.apache.org/proper/commons-lang/ Erişim Tarihi: 07.05.2019.

Anonymous. 2019e Web Sitesti: https://stackabuse.com/hierarchical-clustering-with-python-and-scikit-learn/ Erişim Tarihi: 07.05.2019.

A.J. Offutt, G. Rothermel, and C. Zapf. 1993. An Experimental Evaluation of Selective Mutation. Proc. 15th International Conf. Software Eng., pp. 100-107.

A.J. Offutt, A. Lee, G. Rothermel, R.H. Untch, and C. Zapf. 1996. An Experimental Determination of Sufficient Mutant Operators. ACM Trans. Software Eng. and Methodology, vol. 5, no. 2, pp. 99- 118.

A.J. Offutt, J. Pan. 1997. automatically detecting equivalent mutants and infeasible Paths. Softw. Test. Verif. Reliab. 7(3) pp. 65–192.

A. J. Offutt, R. H. Untch. 2001. Mutation 2000: Uniting the Orthogonal, Kluwer International Series On Advances in Database Systems, Mutation testing for the new century, Section: Mutation: cost reduction, pp. 34-44.

A.P. Mathur. 1991. Performance, Effectiveness, and Reliability Issues in Software Testing. Proc. Fifth Int'l Computer Software and Applications

Conf., pp. 604-605.

A.P. Mathur and W.E. Wong. 1993. An Empirical Comparison of Mutation and Data Flow Based Test Adequacy Criteria. technical report, Purdue Univ. vol. 4, pp. 9-31

A.S. Namin and J.H. Andrews. 2006. Finding Sufficient Mutation Operators via Variable Reduction. Proc. Second Workshop Mutation Analysis, p. 5.

A.S. Namin and J.H. Andrews, “On Sufficiency of Mutants,” Proc. 29th Int’l Conf. Software Eng., pp. 73-74, May 2007.

A.S. Namin, J.H. Andrews, and D.J. Murdoch. 2008. Sufficient Mutation Operators for Measuring Test Effectiveness. Proc. 30th Int’l Conf. Software Eng., pp. 351-360.

A.T. Acree. 1980. On Mutation. PhD thesis, Georgia Inst. Of Technology.

B. H. Smith and L. Williams. 2007. An Empirical Evaluation of the MuJava Mutation Operators. Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION 2007. TAICPART-MUTATION 2007, pp. 193-202.

Chang-ai Sun, Feifei Xue, Huai Liu, Xiangyu Zhang. 2017. A path-aware approach to mutant reduction in mutation testing", Information and Software Technology, Volume 81. Pages 65-81.

Delgado-Pérez, Pedro & M. Rose, Louis & Medina-Bulo, Inmaculada. 2018. Coverage-based quality metric of mutation operators for test suite improvement. Software Quality Journal. 10.1007/s11219-018-9425-7.

Do H, Rothermel G, Kinneer A. 2004. Empirical studies of test case prioritization in a junit testing environment. Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE 2004). IEEE Computer Press: Silver Spring, MD, pp. 113–124.

Dunwei Gong, Gongjie Zhang, Xiangjuan Yao, Fanlin Meng. 2017. Mutant reduction based on dominance relation for weak mutation testing. Information and Software Technology, Volume 81, pp 82-96.

E. F. Barbosa, J. C. Maldonado, and A. M. R. Vincenzi. 2001. Toward the determination of sufficient mutant operators for c. Software: Testing, Verification and Reliability, 11(2):113–136.

Elbaum SG, Malishevsky AG, Rothermel G. 2000. Prioritizing test cases for regression testing. Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2000). ACM Press: New York, 102–112.

E.S. Mresa and L. Bottaci. 1999. Efficiency of Mutation Operators and Selective Mutation Strategies: An Empirical Study. Software Testing, Verification, and Reliability, vol. 9, no. 4, pp. 205-232.

Farmeena Khan and Mohd. Ehmer Khan. 2012. A Comparative Study of White Box, Black Box and Grey Box Testing Techniques. Int. J. Adv. Comput. Sci. Appl, 3(6).

G. J. Myers, T. Badgett, T. M. Thomas, and C. Sandler. 2004. The Art of Software Testing. John Wiley & Sons.

Gregg Rothermel vd. 2001. Prioritizing Test Cases for Regression Testing. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 27, NO. 10.

H. Agrawal, R.A. DeMillo, B. Hathaway, W. Hsu, W. Hsu, E.W. Krauser, R.J. Martin, A.P. Mathur, and E. Spafford. 1989. Design of Mutant Operators for the C Programming Language. Technical Report SERC-TR-41-P, Purdue Univ.

H. Do and G. Rothermel. 2006. On the use of mutation faults in empirical assessments of test case prioritization techniques. IEEE TSE, 32(9):733– 752.

Jia, Y., & Harman, M. (2010). An analysis and survey of the development of mutation testing. IEEE transactions on software engineering, 37(5), 649-678.

J. H. Andrews, L. C. Briand, and Y. Labiche. 2005. Is mutation an appropriate tool for testing experiments. In Proc. ICSE, pp. 402–411.

K.N. King and A.J. Offutt. 1991. A Fortran Language System for Mutation-Based Software Testing. Software: Practice and Experience, vol. 21, no. 7, pp. 685-718.

Leon D, Podgurski A. 2003. A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases. Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE 2003). IEEE Computer Press: Silver Spring, MD, 442–456.

Li Z, Harman M, Hierons RM. 2007. Search algorithms for regression test case prioritization. IEEE Transactions on Software Engineering; 33(4):225–237.

Madeyski, L., Orzeszyna, W., Torkar, R., & Jozala, M. (2013). Overcoming the equivalent mutant problem: A systematic literature review and a comparative experiment of second order mutation. IEEE Transactions on Software Engineering, 40(1), 23-42.

Mathur A. P. 1994. Mutation Testing. In Encyclopedia of Software Engineering, Marciniak J.(ed.). pp. 707–713.

M. Kintis, M. Papadakis, N. Malevris. 2010. evaluating mutation testing alternatives: a collateral experiment. in: Proceedings of the 2010 Asia Pacific Software Engineering Conference (APSEC ’10). pp.300–309.

Mohd. Ehmer Khan. 2011. Different Approaches to Black Box Testing Technique for Finding Errors. International Journal of Software Engineering & Applications (IJSEA), Vol.2, No.4. pp. 31.

M. Papadakis and N. Malevris. 2010. An empirical evaluation of the first and second order mutation testing strategies. in Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops, ser. ICSTW ’10. IEEE Computer Society. pp. 90–99.

M. Polo, M. Piattini, and I. Garcia-Rodriguez. 2008. Decreasing the Cost of Mutation Testing with Second-Order Mutants. Software Testing, Verification, and Reliability, vol. 19, no. 2. pp. 111-131.

Offutt, A. J. and W. M. Craft. 1994. Using compiler optimization techniques to detect equivalent mutants. Softw. Test., Verif. Reliab. pp. 131–154.

R.M. Hierons, M. Harman, S. Danicic. 1999. Using program slicing to assist in the detection of equivalent mutants, Softw. Test. Verif. Reliab. 9(4) 233–262.

S. Hussain. 2008. Mutation Clustering. Master’s thesis, King’s College London.

Srivastava A, Thiagarajan J. 2002. Effectively prioritizing tests in development environment. In ACM SIGSOFT Software Engineering Notes (Vol. 27, No. 4, pp. 97-106). ACM

Srinivas Nidhra and Jagruthi Dondeti. 2012. Black box and white box testing techniques –a literature review. International Journal of Embedded Systems and Applications Vol.2, No.2. Testing and Analysis (ISSTA 2002). ACM Press: New York, pp. 97–106.

S. Yoo and M. Harman. 2012. Regression testing minimization, selection and

prioritization: a survey. Software Testing, Verification and Reliability. 22(2), 7-120

T.A. Budd.1980. Mutation Analysis of Program Test Data. PhD thesis, Yale Univ.

W. B. Langdon, M. Harman and Y. Jia. 2009. Multi Objective Higher Order mutation Testing with Genetic Programming. 2009 Testing: Academic and Industrial Conference - Practice and Research Techniques, Windsor, pp. 21-29.

W.E. Wong. 1993. On Mutation and Data Flow. PhD thesis, Purdue Univ.

W.E. Wong and A.P. Mathur. 1995. Reducing the Cost of Mutation Testing: An Empirical Study. J. Systems and Software, vol. 31, no. 3, pp. 185-196.

Wong WE, Horgan JR, London S, Mathur AP. 1998. Effect of test set minimization on fault detection effectiveness. Software Practice and Experience; 28(4):347–369.

Y. Jia and M. Harman. 2008. Constructing Subtle Faults Using Higher Order Mutation Testing. Proc. Eighth Int'l Working Conf. Source Code Analysis and Manipulation, pp. 249-258.

Yong Liu, Zheng Li, Ruilian Zhao, Pei Gong. 2018. An optimal mutation execution strategy for cost reduction of mutation-based fault localization. Information Sciences, Volume 422, Pages 572-596.

Yu-Seung Ma and Sang-Woon Kim. 2016. Mutation Testing Cost Reduction by Clustering Overlapped Mutants. Journal of Systems and Software, Volume 115, Pages 18-30.

Zhang, Lingming & Gligoric, Milos & Marinov, Darko & Khurshid, Sarfraz. 2013. Operator-based and random mutant selection: Better together. 2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013 - Proceedings. 92-102. 10.1109/ASE.2013. 6693070.

ÖZGEŞMİŞ

Adı Soyadı	: Mohamed Abdillahi WARSAME
Doğum Yeri	: Hargeisa, Somali
Doğum Tarihi	: 14.11.1989
Medeni Hali	: Bekar
Yabancı Dili	: İngilizce, Türkçe

Eğitim Durumu (Kurum ve Yıl)

Lise	: Tima-ade lisesi (2009)
Lisans	: Alpha Üniversitesi Bilgi ve İletişim Teknolojisi Fakültesi, Bilgisayar Bilimi Bölümü (2012)
Yüksek Lisans	: Ankara Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı (Eylül 2019)