T.C.
İSTANBUL KÜLTÜR ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

PLATFORMLAR ARASI KOD KLON TESPİTİ

YÜKSEK LİSANS TEZİ

TAYFUN TUNÇ
0501020043

Anabilim Dalı: Bilgisayar Mühendisliği
Program: Bilgisayar Mühendisliği

Tez Danışmanı: Doç. Dr. Akhan Akbulut

OCAK 2023

T.C.
İSTANBUL KÜLTÜR ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

PLATFORMLAR ARASI KOD KLON TESPİTİ

YÜKSEK LİSANS TEZİ

TAYFUN TUNÇ
0501020043

Anabilim Dalı: Bilgisayar Mühendisliği
Program: Bilgisayar Mühendisliği

Tez Danışmanı: Doç. Dr. Akhan Akbulut
Jüri Üyeleri: Prof. Dr. Özgür Koray Şahingöz
Dr. Öğr. Üyesi Fatma Patlar Akbulut

OCAK 2023

ÖNSÖZ

“Platformlar Arası Kod Klon Tespiti” adlı tez çalışmamda yönlendirici, bilgi verici, ufuk açan tutumu ile araştırmalarıma olan katkılarından ve desteğini sağlayan değerli tez danışmanım Doç. Dr. Akhan AKBULUT’a çok teşekkür ederim. Her zaman, her durumda koşulsuz sevgilerini ve desteklerini esirgemeyen sevgili eşime ve ailesine, anneme, babama ve kardeşlerime teşekkür ederim.

Ocak 2023 Tayfun TUNÇ

İÇİNDEKİLER

TABLO LİSTESİ

ŞEKİL LİSTESİ

KISALTMALAR VE TERİMLER

Http	: The Hypertext Transfer Protocol - Veri transfer protokolü
AST	: Abstract Syntax Tree - Soyut Sözdizimi Ağacı
PDG	: Program Dependence Graph - Program Bağımlılık Grafiği
CFG	: Control Flow Graph - Kontrol Akış Grafikleri
GitHub	: Git adlı bir sürüm kontrol sistemini barındıran bulut tabanlı bir hizmettir
LSTM	: Long Short-Term Memory - Uzun-Kısa Süreli Bellek
GRU	: Gated Recurrent Unit - Tekrarlayan Geçitli Birim
RvNN	: Recursive Neural Network - Yinelemeli Sinir Ağı
RNN	: Recurrent Neural Network - Tekrarlayan Sinir Ağı
SCDetector	: Semantic Clone Detector - Semantik Klon Tespit Aracı
AGC	: AtCoder Grand Contest - AtCoder Büyük Yarışması
ARC	: AtCoder Regular Contest - AtCoder Standart Yarışması
ABC	: AtCoder Beginner Contest - AtCoder Acemi Yarışması
IDE	: Integrated Development Environment - Etkileşimli Geliştirme Ortamı

Üniversite	:	T.C. İstanbul Kültür Üniversitesi
Enstitü	:	Lisansüstü Eğitim Enstitüsü
Anabilim Dalı	:	Bilgisayar Mühendisliği
Program	:	Bilgisayar Mühendisliği
Tez Danışmanı	:	Doç. Dr. Akhan AKBULUT
Tez Türü ve Tarihi	:	Yüksek Lisans – Ocak 2023

ÖZET

PLATFORMLAR ARASI KOD KLON TESPİTİ

Günümüzün taleplerine ayak uydurmak için yazılım mimarisi sürekli olarak geliştirilmektedir. En modern mimari yöntemlerden biri olmasına rağmen, mikroservis mimarisini uygulamanın zorlukları vardır. Mikroservis mimarisinin çok dilli yapısı organizasyon kolaylığı sağlarken, kod klonlarının algılanmasını da zorlaştırır. Kod klonları, bir yazılım sistemindeki, sistemi korumak için gereken bakım zamanını ve ihtiyaç duyduğu kaynağı artıran, yinelenen kod parçalarını ifade eder. Yazılım geliştirme için kod klonlarının hem olumlu hem de olumsuz etkileri olabilir. Bu nedenle, herhangi bir kod gözden geçirme ve yeniden düzenleme öncesi klonların tanımlanması gerekir. Çok dilli mikroservislerde, platformlar arasında kod klonlarını tespit etmek, onları aynı platformda bulmaktan daha zorlu bir iştir. Bu araştırma, JavaScript, C, C# ve Java ile geliştirilen yazılım bileşenlerinin, birbirinin kod klonları olma derecesini analiz etmek için bir metodoloji sunmaktadır. Spesifik olarak, kodların ne kadar benzer olduğunu belirlemek için Kosinüs Benzerliği, Öklid Uzaklığı ve Manhattan Uzaklığı ölçümlerini kullandık. Deney yoluyla, https://atcoder.jp sitesinden kod kopyalarından oluşan bir veri kümesi topladık ve çeşitli yaklaşımlar için deneysel eşikler belirledik. Deneylerimiz, platformlar arası kod klonlarını tespit etmede en etkili yaklaşımın Manhattan Uzaklığı olduğunu (%91,59), bunu Öklid Uzaklığı (%91,08) ve son olarak da Kosinüs Benzerliği'nin (%72,83) izlediğini

gösteriyor. Aynı platformda kod klonlarını tespit etmedeki başarı oranı ise Kosinüs Benzerliği %94,73, Öklid Uzaklığı %91,77 ve Manhattan Uzaklığı %91,15'tir. Bu tekniğin, yazılım geliştirme ekiplerinin çeşitli teknolojiler kullandığı ortamlarda ortaya çıkabilecek platformlar arası benzer kod örneklerini belirlemek için kullanılabileceği tespit edilmiştir.

Anahtar Sözcükler mikroservis, mikroservis mimarisi, kod klon tespiti, çok dilli programlama

University	:	T.C. İstanbul Kültür University
Institute	:	Institute of Graduate Studies
Department	:	Computer Engineering
Program	:	Computer Engineering
Thesis Advisor	:	Assoc. Prof. Akhan AKBULUT
Degree Awarded And Date	:	MA –Jan 2023

ABSTRACT

To keep up with the demands of today, software architecture is continuously being refined. Although it is the most modern architectural method, adopting a microservice architecture is not without its difficulties. While the polyglot structure of the microservice Architecture provides organizational ease, it also makes detecting code clones more difficult. Code clones refer to duplicate fragments of code in a software system, which increases the time and effort needed to maintain the system and the amount of resources it needs. For software development, code clones can have both positive and negative effects. Therefore, clones need to be identified prior to any sort of refactoring or elimination. In multilingual microservices, recognizing code clones across platforms is a more challenging task than finding them on the same platform. This research presents a methodology for analyzing the degree to which software components developed with JavaScript, C, C#, and Java are code clones of one another. Specifically, we utilized the Cosine Similarity, Euclidean Distance, and Manhattan Distance measures to determine how similar codes were. Through experimentation, we have amassed a dataset consisting of code copies from the site https://atcoder.jp and identified empirical thresholds for various approaches. Our research indicates that Manhattan Distance is the most effective approach for detecting cross-platform code clones (91.59%), followed by Euclidean Distance (91.08%), and finally Cosine Similarity (72.83%). The success rate in detecting code clones on the same platform is Cosine Similarity 94.73%, Euclidean Distance 91.77%, and Manhattan Distance 91.15%. It’s been established that this technique can be used to

identify instances of code duplication across platforms, which might arise in settings where development teams employ diverse technology stacks.

Keywords: microservice, microservice architecture, code clone detection, polyglot, multilingual programming

1. GİRİŞ

Açık kaynak kod grupları, son on yılda güçlü yazılımlar geliştirerek açık kaynak yazılımların büyümesine katkıda bulundu. Özellikle iş birlikleriyle üretilen uygulamaların olgunluk ve kalitesinin artması bu büyümeye önemli katkı sağlıyor. 2022'nin en çok okunan açık kaynak yayınlarından biri olan GitHub'ın Octoverse Raporu [1] bu genişlemeyi vurguluyor. Analiz, geliştiricilerin yalnızca 2022'de GitHub'da 52 milyon yeni açık kaynak projesi oluşturduğunu ve mevcut açık kaynak projelerine 413 milyondan fazla ekleme yaptığını gösteriyor. Fortune 100 şirketlerinin %30'undan fazlasının artık stratejileri açık kaynakta koordine etmek için açık kaynak program ofisleri var. 94 milyondan fazla geliştirici, şu anda dünyadaki yazılımların %90'ından fazlasının temelini oluşturan GitHub ve açık kaynak kullanıyor.

Günümüz yazılım dünyasında, çok sayıda onaylı yazılım mimarisi bulunmaktadır. Projenin gereksinimlerine uygun olarak monolitik, dağıtık, n-katmanlı, mikro çekirdek ve hibrit seçeneklerden en yaygın trendler seçilir. Mikroservisler, günümüz standartlarına göre, mevcut seçeneklerin özelliklerini analiz eden ve seçeneklerin özelliklerini değerlendirdikten sonra dağıtık bir mimariyi benimsemeye karar veren yazılım mimarları için tercih edilen mimari stil olmalıdır. Mikroservisler yaygınlaştırma ve yükseltme kolaylığı, yüksek ölçeklenebilirlik, pazara çıkma süresinde çeviklik, yüksek bakım ve dağıtım otomasyonu, azaltılmış güvenlik tehditleri ve güvenilirliği, optimize edilmiş geliştirme süresi ve harcamaları ile göz önüne alındığında, çok dilli yapıya sahip yazılım geliştirme ekipleri için ön plana çıkmaktadır.

Programlama toplulukları, büyük projeler için tek bir geliştirme platformuna güvenmenin zararlı olduğunu belirtiyor. Güvenlik ve performans için Java ve C# gibi üst düzey programlama dillerine ağırlık verilirken, işlem tabanlı modüller için JavaScript, veri bilimi ve görüntü işleme için Python programlama dili tercih edilir. Bu, geliştiricilerin farklı programlama dilleri arasında geçiş yapma konusunda giderek daha rahat olduklarını gösteriyor. Günümüzde giderek daha fazla sayıda geliştirici, bir ürün oluştururken en etkili çözümleri entegre etmek amacıyla kod yazmak için birden fazla programlama dili kullanıyor. Doğal olarak bu, Yazılım Kalite Güvencesi yöntemlerinin işleyişini zorlaştırmaktadır. Böyle bir durumda, kod klonlarını tespit etmek daha zor hale gelmektedir.

Bir kod parçasını kopyalamak, tekrar kullanmak veya farklı bir konumda değiştirilerek kullanılması yazılım geliştirme de yaygın kullanılan bir pratiktir [2]. Bir yazılım projesinde klonlanmış koda sahip olmanın iki önemli olumsuz yanı vardır: (i) klonlanmış kodda yapılan tutarsız değişiklikler, sorunlara ve hatalı program davranışına neden olabilir, bu da (ii) bakım masraflarını artırır. Ayrıca Yazılım Kalite Güvence prosedürlerindeki kod klonları, bu amaç için tasarlanmış araçlar kullanılarak çıkarılmaktadır. Burada sunulan araştırma, yukarıda belirtilen sorunun çözümüne üç farklı katkı sağlamaktadır.

i. İki kod parçasının birbiriyle ne kadar yakından ilişkili olduğunu belirlemek için Kosinüs Benzerliği, Öklid Uzaklığı ve Manhattan Uzaklığı yaklaşımlarını inceledik ve karşılaştırdık.
ii. Java, JavaScript, C ve C# programlama dillerinde yazılmış 620 kod parçasından 34 yazılım metriğinin çıkarılmasıyla, platformlar arası kod klonu tespitinde kullanılmak üzere gerçek bir veri kümesi oluşturduk.
iii. Deneysel araştırmamızın bir parçası olarak, aynı kod parçaları ile çapraz platformlar arasındaki benzerlik derecesini analiz ettik. Potansiyel kod klonlarını belirlemek için çeşitli programlama dillerinin birlikte çalışabilirliğini de araştırdık.

1.1.Problem Tanımı

Teknolojinin gelişimi ile ortaya çıkan yeni yazılım mimarilerinde artık birçok yazılım geliştirici ortak bir proje üzerinde kod geliştirebilme imkanına sahiptir. Projelerin doğası gereği programlar bazen tek bir programlama dili ile geliştirilirken bazen de birden fazla programlama dilleri kullanılabilmektedir. Bir sorunun çözümü için yazılan bir kod parçası farklı programlama dillerinde tekrar tekrar yazılmaktadır. Özellikle mikroservis mimarilerinin artması ve yazılım projelerinde kullanılan programlama dillerinin çeşitlenmesi ile kod tekrarlarının bakım maliyetleri de artmaktadır. Kod klon tespiti ile bakım maliyetleri azaltılabilir. Açık kaynak kodlu yazılımların çoğaltılması sırasında ortaya çıkan hatalı kod parçasının klonlanarak kullanılması ve hatanın çoğaltılması problemleri ile karşı karşıya kalınmaktadır. Kod klon tespiti ile çoğaltılan hatalı kodlar tespit edilebilecek ve hatalı süreçlerin düzeltilmesi sağlanabilecektir. Kaynak kodların yasal olmayan şekilde çoğaltılması, intihal veya telif hakkı ihlalleri için kod klon tespiti kullanılır.

1.2.Literatüre Katkıları

Monolitik uygulamalardan mikroservis mimarisine yapılacak dönüşümlerin bakım ve geliştirme maliyetleri düşürülmesine katkı sağlar. Aynı platform içerisindeki kod klon tespiti yapılarak bakım, onarım ve geliştirme maliyetleri azaltılır. Aynı platform içerisinde belirlenen hatalı klonların önüne geçilerek oluşabilecek zararlar önceden tespit edilir. Platformlar arası kodların çoğaltılmasıyla ortaya çıkan karmaşık yazılımlarda bakım, onarım ve geliştirme maliyetleri azaltılır. Kaynak kodların yasal olmayan şekilde çoğaltılması engellenir, intihal veya telif hakkı ihlallerini tespit etmek için faydalanılır. Farklı platformlardaki hatalı kod versiyonlarının dağıtılmasının önüne geçilir. Kodları gözden geçirip iyileştirme alanlarını belirlenmesinde kullanılır. Farklı klon türlerindeki kod klonlarını tespit etmede kullanılır. Metinsel olarak farklı olan ancak işlevsel olarak aynı işlevselliği sağlayan klonların tespitleri de yapılabilir.

1.3.Motivasyon

Mikroservis mimarisine dönüşüm projelerinde ortaya çıkan mevcut yazılımların tekrar gözden geçirilmesi ve küçük program parçaları ile tekrar yazılması, projelerin yönetimi noktasında zorluklar çıkartabilmektedir. Kaliteden ödün vermeden projenin zaman, bütçe ve diğer kriterlerine uygun bir şekilde geliştirilebilmesi için kod klonların tespiti, yönetilmesi ve takibi önemli bir yer teşkil etmektedir. Kod klon tespiti yapan bazı uygulamalar mevcut ancak bu araştırmayı yapmaktaki motivasyon kaynağı farklı platformlardaki kod klonları üzerine çalışmalar yaparak mikroservis yapılarında ve çeşitli programlama dilleri kullanan projelerde kod klon tespitlerine fayda sağlamaktır. Deneylerimizdeki en önemli sorunlardan bir tanesi Bölüm 2.3'te detaylı bahsettiğimiz kod klon türlerinden anlamsal olarak benzer kod klon türlerinin tespitlerindeki zorluklar. Bir diğer sorun ise çapraz programlama dillerindeki kod benzerliklerini bulmak için yapılan çalışmalarda programlama dillerinin yapıları gereği ortaya çıkan farklılıkların tespit edilmesidir. Programlama dillerinin farklı amaçlarla tasarlanmış olmasından kaynaklı farklı kabiliyetler sunan özellikleri kod klon tespitlerini zorlaştırmaktadır. Ayrıca yazılım geliştiricilerin aynı problem için bile farklı algoritmalar ile benzer kod parçaları yazması da deneylerimizdeki bir diğer zorluğu ortaya koymaktadır. Hipotezlerimizden bazılarını da şöyle sıralayabilirim Kosinüs Benzerliği, Öklid Uzaklığı ve Manhattan Uzaklığı teknikleri Ngram ve Jeccard tekniklerine oranla daha başarılı sonuçlar ortaya koyacaktır. Kodlardaki metinsel değerlerin sayısal metriklere çevrilmesinden ziyade anlamsal kavramların

sayısallaştırılıp metrikler elde edilmesi ile benzerlik oranları artabilir. Metinsel olarak benzer olan kod klon türleri diğer kod klon türlerine göre daha kolay tespit edilebilir. Özellikle aynı kökenden gelen programlama dilleri arasında çapraz klon tespitleri daha yüksek oranda başarılı olabilirken farklı amaçlar için tasarlanmış programlama dilleri için bu oran düşebilir. Veri kümeleri oluşturulurken problem çözümleri özellikle satır sayısı olarak düşük olan verilerde kod benzerlik başarı oranları düşük tutarlılıkta sonuçlanabilir. Bu tür hipotezlerimizin ortaya çıkmasında aşağıdaki araştırma sorularımızın (AS) katkısı olmuştur.

AS-1: Kod klonlarının benzerliğinin hesaplanmasında hangi yöntemler kullanılabilir?

AS-2: Kod klon tespitinde hangi yazılım metrikleri önemli rol oynar?

AS-3: Hangi platformlar ya da programlama dilleri çapraz kod klon tespiti için daha uyumludur?

AS-4: Hangi veri kümeleri çapraz kod klon tespiti için daha uygundur?

1.4.Tezin Organizasyonu

Bu tez yedi bölümden oluşmaktadır. Birinci bölümde, problem tanımı yapılmış, yapılan çalışma tanıtılmış, literatüre katkısından söz edilmiş, motivasyon kaynağı ve amacından bahsedilmiştir. İkinci bölümde, tez çalışmasının konusu olan kod klonlarından, platformlar arası kodların gelişiminin kullanım alanları olan mikroservis mimarilerinden, kod klon türlerinden, kod klon tespitinden bahsedilmiştir. Tezin üçüncü bölümünde, literatür araştırmasından bahsedilmiştir. Dördüncü bölümde, kullanılan yöntemlerden, araştırmalarda kullanılan uygulamalardan, seçilen problemlerden ve özdeş kodların bulunması için kullanılan hesaplama tekniklerinden bahsedilmiştir. Kullanılan teknolojilerden ve programlama dillerinden bahsedilmiştir. Beşinci bölümde, araştırmalar sırasında yapılan deneylerden, analizlerden, yapılan testlerden ve bulunan bulgulardan bahsedilmiştir. Altıncı bölümde, araştırmalarımızda ortaya çıkan riskler, engeller ve gelecekteki yönlendirmelerden bahsedilmiştir. Yedinci bölümde ise, yapılan çalışmanın ürettiği çıktılar üzerinden sonuçlar ele alınmıştır.

2. ÖN BİLGİLER

Bu bölümde platformlar arası kod klon tespiti konusu üzerine literatürde tespit ettiğimiz yaklaşımlardan bahsedilmiştir ve konunun kapsadığı terminoloji hakkında açıklamalar yapılmıştır. Platformlar arası kod klonları araştırılırken farklı programlama dilleri ile geliştirilen mikroservis mimarisi üzerine kurgulanmış projeler incelenmiştir. Farklı kod klon türleri ele alınmıştır. Bu klon türleri üzerine yapılan kod özdeşlerini tespit etme teknikleri araştırılmıştır. Kod klon tekniklerinin genel yapıları incelenmiştir. Bu bölümde bu incelemeler sırasında edinilen bilgi ve teknik tanımlar aktarılmıştır.

2.1.Mikroservis Mimarisi

Mikroservis bileşenleri; monolitik bir sistemin, her biri bağımsız olarak çalışabilen ve açık protokoller (örneğin http) vasıtasıyla birbiri ile iletişim kurabilen küçük servislere ayrılması olarak ifade edilir [3]. Mikroservis mimarilerinin temel amacı ve en önemli faydalarından biri bağımsız çalışabilmesi ve geliştirilebilmesidir. Bu sayede yazılım geliştiriciler hizmet sunduğu kullanıcılarına minimum düzeyde etkilenerek yazılım geliştirmelerini sağlayarak, büyük projelerdeki karmaşıklığın önüne geçer. Sonuç olarak yazılım geliştiriciler projelerinde diğer ekiplerden bağımsız hazır uygulamaları da entegre edebilirler ve daha hızlı geliştirmeler yaparak yayına bağımsız bir şekilde alabilme kabiliyetine sahip olurlar. Yazılım ekiplerinin ürünlerini yayına alma noktasındaki iletişim karmaşasını basitleştirmekte ve sürdürülebilirliğe katkı sağlamaktadır.

2.2.Kod Klonları

Bir yazılım projesinde bulunan benzer veya birebir özdeş kod parçalarına kod klon denir [4] . Yazılım geliştiriciler zaman zaman bilinçli olarak veya üzerinde çalıştıkları projelerin çözümleri gereği benzer kodlar yazmaktadır. Bazen projelerdeki zaman ve bütçe kısıtlarından dolayı hızlı geliştirme yapmaları gerektiğinde daha önce yazdıkları kodları kopyalayıp yapıştırarak yeni projelerinde de kullanarak kod klonları oluşturmaktadırlar. Bir projede farklı farklı yazılım ekipleri çalışabilir ve bu ekiplerde aynı çözümü farklı yazılımcılar yeniden geliştiriyor olabilirler bu da yine kod klonların ortaya çıkmasının başlıca nedenlerindendir.

2.3.Kod Klon Türleri

Farklı araştırmaların kategorizasyonu değişse de genellikle araştırmalarda kod klonlarının dört farklı başlıkta sınıflandırıldığı görülmektedir.

Kategori 1 (Birebir özdeş): Boşluk ve yorumlar dışında iki özdeş kod parçasına sahip kod çiftleri.

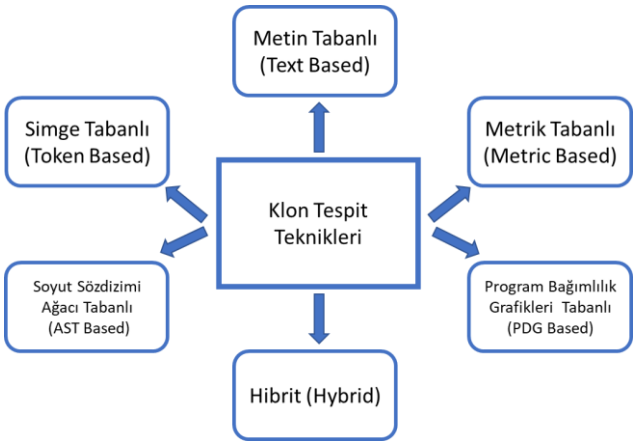
Kategori 2 (Yeniden adlandırılmış): Değişken adı, tip adı ve fonksiyon adı dışında iki özdeş kod parçasına sahip kod çiftleri.

Kategori 3 (Neredeyse özdeş): Eklenmiş veya kaldırılmış bir dizi ifadeye sahip olan veya farklı tanımlayıcılar, metinler, tipler, boşluklar, şablonlar ve yorumlar kullanan ancak yine de benzer olan kod çiftleri.

Kategori 4 (Anlamsal olarak benzer): Metinsel olarak benzer olmayan ancak anlamsal olarak benzer olan kod çiftleri [5].

2.4.Kod Klon Tespiti

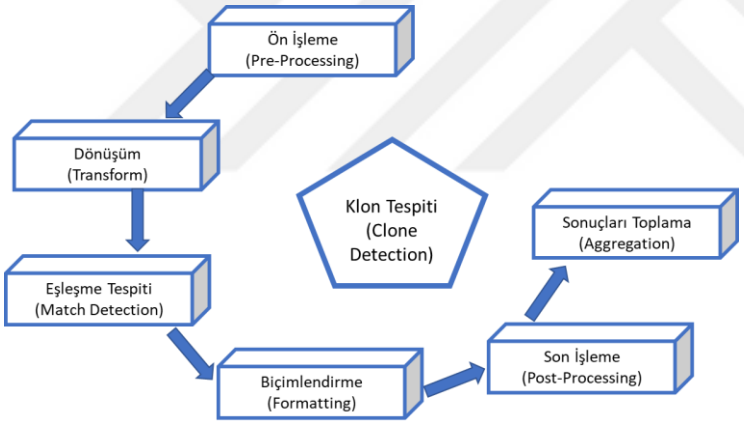
Kod klon tespiti yazılım projeleri içerisindeki kaynak kodlarda özdeş veya benzer kod parçalarının bulunmasına denir [6]. Kodların metinsel olarak veya fonksiyonel olarak kopyalanarak çoğaltılması ve çoğaltıldığı noktalarda bunların tespit edilmesi süreci kod klon tespiti olarak adlandırılır. Literatürde kullanılan genel hatları ile ortaya konmuş kod klon tespit teknikleri Şekil 2.1'de gösterilmiştir.



Şekil 2.1 Klon Tespit Teknikleri

Manuel olarak kod klon tespiti yapılabildiği gibi otomatik olarak kod klon tespiti yapan araçlar da geliştirilmiştir. SourcererCC [7], DECKARD [8], CDLH [9],

DeepSim [10] ve ASTNN [11]. Kod klon tespitinde kullanılan teknikler genel olarak Şekil 2.1'de verilmiştir. Kod klon tespiti yazılım bakımları ve gelişimlerinde önemli bir konu teşkil etmektedir. Klon tespit kütüphaneleri için olası adayları tespit etmede yeni eklemeler yapmak [12] amacıyla tanımlanmış klonların bazılarını kaynak kodundan çıkarmak [13], programların anlaşılır olmasına yardımcı olmak için [14], zararlı yazılımların tespit edilmesinde [15], intihal veya telif hakkı ihlallerinin tespitinde, bağlama bağlı olarak tutarsızlıkların tespitinde [16], klon tabanlı hata tespitinin uygulanabilirliği [17] gibi konularda kod klonlardan faydalanılabilir. Klon işleminin zararlı olma olasılığı klonlanacak kodun bazı özellikleri ile ilgili olabilir [18] ve kodları gözden geçirip iyileştirme alanlarını belirlenmesinde kullanılmaktadır [19] [20] [21]. Kod klon tespitlerinde genel olarak izlenen yöntemler benzerlik göstermektedir [22] [23]. Kod klon tespitinde genel olarak izlenen süreçler Şekil 2.2'de gösterilmiştir.



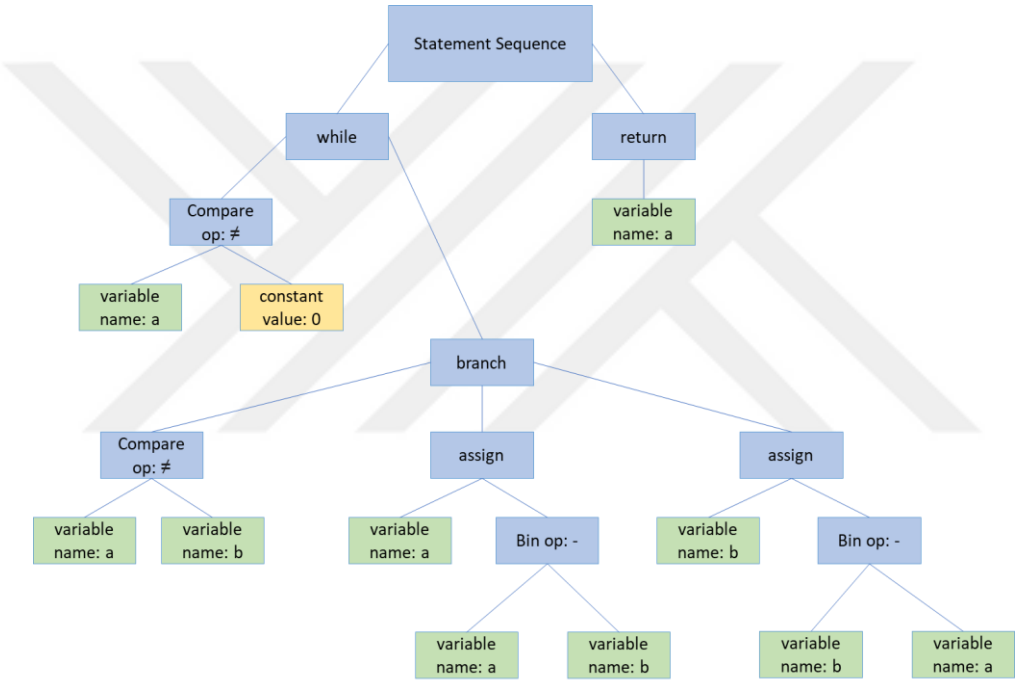
Şekil 2.2 Klon Tespit Adımları

Ön İşleme: Yorum satırları gibi işlevsel kod dışındaki metinler temizlenir. Dönüşüm: Metrikler ile soyutlama, Soyut Sözdizimi Ağacı (AST - Abstract Syntax Tree) [24], Program Bağımlılık Grafikleri (PDG - Program Dependence Graph) [25] veya Kontrol Akış Grafikleri (CFG - Control Flow Graph) [26] gibi kaynak kodların farklı türde soyutlamaları çıkartılır. Eşleşme Tespiti: İki kaynak kodu arasındaki mesafe hesaplanır, bu mesafe belirlenen bir eşik değerine ulaştığında kod klon tespiti yapılmış olur. AST bir programlama dilinde yazılmış kaynak kodun soyut söz dizimsel yapısının bir ağaç temsilidir. Şekil 2.3'te basit bir döngü ve bir koşul içeren kod örneği

verilmiştir. Bu kod örneğinin AST gösterimi de Şekil 2.4'te verildiği gibi ortaya çıkmaktadır.

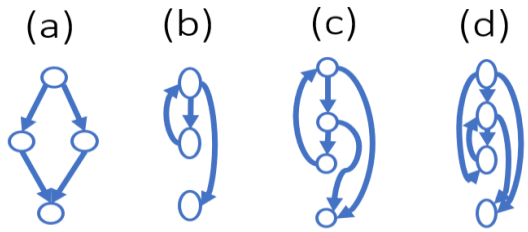
```
while b ≠ 0
 if a > b
   a := a – b
 else
   b := b – a
return a
```

Şekil 2.3 Kod Örneği 1. Bir döngü ve bir koşul içeren kod örneği.



Şekil 2.4 Kod Örneği 1 için oluşturulmuş AST

PDG hem veri bağımlılığını hem de kontrol bağımlılığını içeren bir grafik gösterimidir. CFG ise yürütülmesi sırasında bir programın içinden geçilebilecek tüm yolların grafik gösterimini kullanan bir temsildir. Şekil 2.5'te CFG Örnekleri verilmiştir. (a) "if-then-else" örneği, (b) "while" döngü örneği, (c) İki çıkış noktası olan ve ortada "if-break" ile tanımlanmış bir döngü örneği, (d) İki giriş noktası olan bir döngü örneği olarak gösterilmektedir.



Şekil 2.5 CFG Örnekleri. (a) “if-then-else”, (b) “while” döngüsü, (c) İki çıkış noktası olan ve ortada “if-break” ile tanımlanmış bir döngü, (d) İki giriş noktası olan bir döngü.

```
7   namespace KodKlonTespiti
8   {
        0 references
9       public class KodKlonTespiti
10      {
            0 references
11          public int A()
12          {
13              int res = 1,i=0;
14              int len = 10, flag = 100;
15              for (; i < len; i++)
16              {
17                  res += i;
18              }
19              if (res > flag)
20              {
21                  res = res - flag;
22              }
23              else
24              {
25                  res = flag - res;
26              }
27              return 0;
28          }
29
```

Şekil 2.6 Kod Örneği 3. C# programlama dili ile yazılmış. Basit döngü ve koşullar içeren Yöntem A.

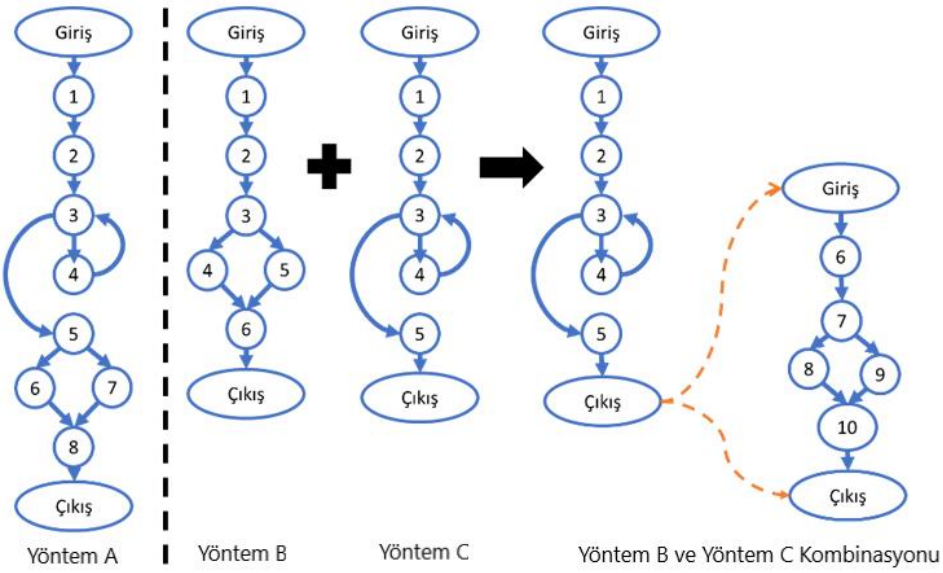
```
7   namespace KodKlonTespiti
8   {
        0 references
9       public class KodKlonTespiti
10      {
            0 references
11          public int A()...
29
            1 reference
30          public int B(int a, int b)
31          {
32              int c;
33              if (a > b)
34              {
35                  c = a - b;
36              }
37              else
38              {
39                  c = b - a;
40              }
41              return c;
42          }
43
```

Şekil 2.7 Kod Örneği 4. C# programlama dili ile yazılmış. Basit bir koşul içeren Yöntem B.

```
7   namespace KodKlonTespiti
8   {
        0 references
9       public class KodKlonTespiti
10      {
            0 references
11          public int A()...
29
            1 reference
30          public int B(int a, int b)...
43
            0 references
44          public int C()
45          {
46              int res = 1, i = 0;
47              int len = 10, flag = 100;
48              while (i<len)
49              {
50                  res += i;
51                  i++;
52              }
53              res = B(res, flag);
54              return 0;
55          }
56      }
57  }
```

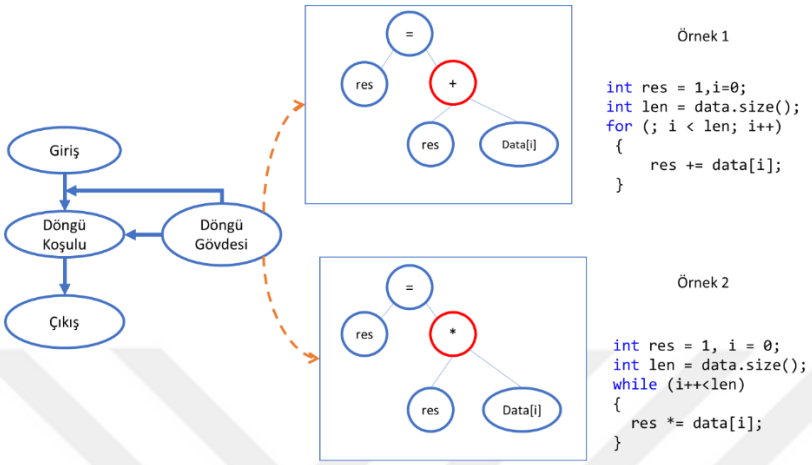
Şekil 2.8 Kod Örneği 5. C# programlama dili ile yazılmış. Basit bir döngü içeren Yöntem C.

Şekil 2.9'da verilen örnekte Şekil 2.6'da verilen Yöntem A, Şekil 2.7'de verilen Yöntem B ve Şekil 2.8'de verilen Yöntem C'nin CFG gösterimi bulunmaktadır. Yöntem B ile Yöntem C'nin CFG'lerini bir araya getirdiğimizde Yöntem A ile aynı işlevselliği sağlamaktadır. Bu sayede kod klon tespiti yapılabilmektedir.



Şekil 2.9 Yöntem A, B ve C'nin CFG gösterimi. Yöntem B ve Yöntem C Kombinasyonu

Şekil 2.10 içerisinde verilen Örnek 1 ve Örnek 2 kod parçaları aynı şekil içerisinde önce AST olarak verilmiştir. Sonra da CFG gösterimleri verilmiştir.



Şekil 2.10 Örnek 1 ve Örnek 2 için AST ve CFG

3. LİTERATÜR TARAMASI

Bu bölümde kod klon tespitleri ile ilgili yapılan diğer çalışmalardan bahsedilmektedir. Kod klon tespiti üzerine literatüre bakıldığında birçok çalışma tespit edilmiştir. Bu çalışmaların çoğu eforunu tek bir programlama dili üzerinde kod klon tespiti yapmaya yönelik veya tek bir program üzerinde bulmaya yönelik yaptığı görülmektedir. Statik analiz, temel fikir olarak programların kodlarını çalıştırmadan statik olarak analiz etme üzerine kuruludur. David ve Yahay 2014 yılında yöntemlerin benzerliklerinin izlerini yakalamak üzerine çalışmalar yapmıştır [27]. İkili sayı sistemi tabanlı verilen büyük kaynak kodlarda statik olarak benzer işlevler bulma üzerine çalışmalar ortaya konmuştur. Kodlardaki benzerlikleri bulurken kaynak kodları statik olarak analiz edip fonksiyonları kendi belirledikleri izlere çevirmişlerdir. Bu izler sayesinde tersine mühendislik teknikleri kullanarak kaynak kodlardaki diğer izleri adreslemek üzerine araştırmalar yapmışlardır. Daniel Perez ve Shigeru Chiba'nın 2019 yılında yayınladığı makalesinde platformlar arası kod klon tespitinde AST kullanımı üzerine çalışmalardan bahsedilmiştir.

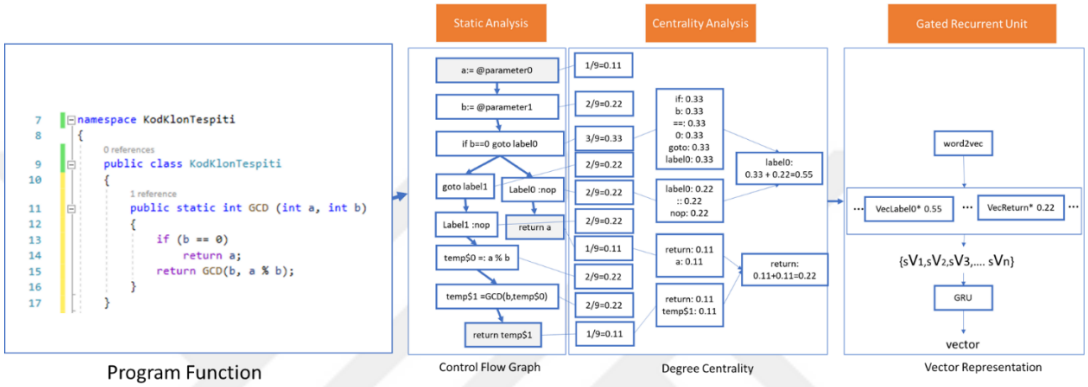
Özellikle mikroservis mimarilerindeki farklı yazılım ekiplerinin farklı programlama dilleri üzerinde yetkinlik sahibi olması üzerine çapraz programlama dilleri üzerinde kod klon tespitlerine odaklanılmıştır. Java ve Python programlama dilleri üzerinde 130 milyonun üzerinde satır kaynak kodu olan projeler incelenmiştir. 576 farklı senaryo ile ilgili kaynak kodlar toplanmıştır. Yapılan çalışmada anahtar nokta olarak jeton tabanlı (token based) vektörlerin oluşturulması olarak vurgulanmıştır. Özellikle doğal dil işleme literatüründe kullanılan tekniklerden biri olan Skip Gram tekniği ile modelleme yaparak veriler oluşturulmuş kod klon tespiti üzerine çalışmalar gerçekleştirilmiştir [28]. Eschweiler, Yakdan ve Gerhards-Padilla 2016 yılında potansiyel klon adaylarını hesaplamak için sayısal özelliklerden faydalanan ve benzerlikleri bulabilmek içinde yapısal özelliklerden faydalanan bir çalışma yürütmüştür. Kaynak kodlardaki benzerliklerin hesaplanmasında kontrol akış grafiklerine dayalı olarak ilerlenmiştir. Hesaplama maliyetlerini azalmak adına sayısal özellikleri temel alan filtrelemeler yaparak süreci hızlandırma çalışmaları yapılmıştır. discovRE olarak adlandırılan bir prototip ortaya çıkartılmıştır. İkili sayma sistemindeki hata tespit araçları arasında hızlı bir araç olduğunu gösterecek akademik çalışmalar ortaya konulmuştur [29]. CCFinder [6], SourcererCC [7] simge tabanlı

(token based) bir teknik ile kod klon tespiti yapmaktadır. Bu teknik özellikle üçüncü tür kod klon tespitlerinde verimli çalışmaktadır. Bu tür teknikler çapraz platformlar arası kod klon tespitleri yapmak için tasarlanmamıştır. Deckard [30], soyut sözdizimi ağacı tabanlı (abstract syntax tree based) bir yaklaşım ile kod klon tespiti yapmaktadır, bu da aynı programlama dili içerisindeki kod klonların tespiti üzerine bir çalışmadır.

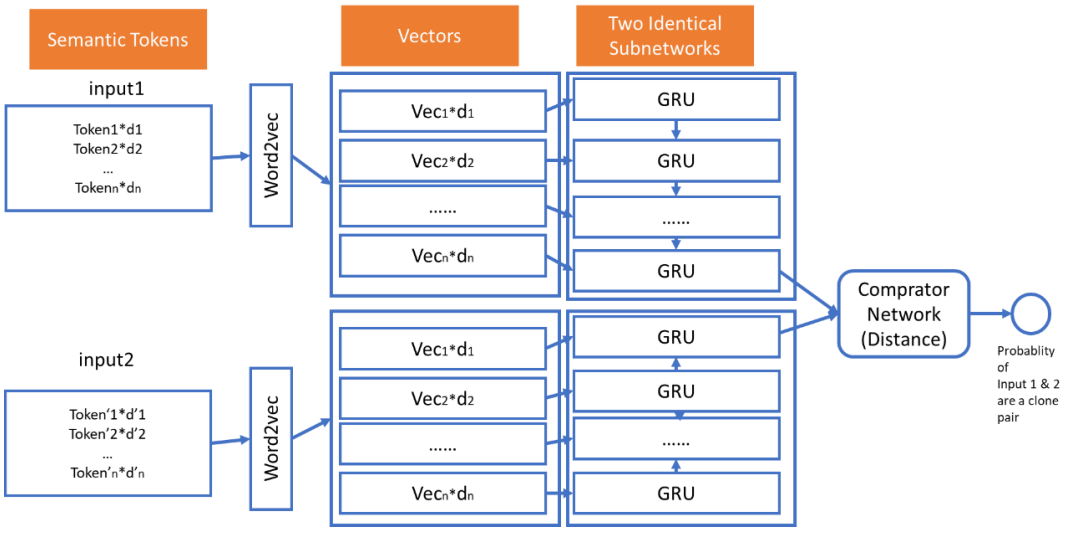
Bazı yaklaşımlarda ise benzerlik gösteren programlama dilleri arasında kod klon tespiti yapmak üzerine çalışmalar gerçekleştirilmiştir. Örneğin C# ile Visual Basic .Net arasında nesneleri ara bir katmana dönüştürerek benzerliklerini tespit etmek üzerine çalışmalar yapılmıştır. Bu tür çalışmalar tamamen birbirinden bağımsız dillerde işe yaramamaktadır. Örneğin Java ile Python programlama dillerindeki kod klon tespitlerinde kullanılamaz. Benzer dillerdeki nesneleri CodeDOM [31] aracını kullanarak ortak bir ara katmana dönüştürüp, bu şekilde benzerliklerini ölçmek üzerine kurgulanmıştır. Shalev ve Partush 2018 yılında makine öğrenmesi tekniğini ikili sayma sistemi üzerinden kod benzerliklerinin tespiti için kullanmıştır [32]. Feng 2016 yılında grafik tabanlı bir yöntem ile platformlar arası ve farklı mimarilerdeki yapılarda hata tespit çalışmaları yapmıştır [33]. Anlamsal bilgiye dayalı çalışmalardaki genel eğilim program bağımlılık grafikleri olarak adlandırılan PDG tabanlı çalışmalardır. PDG, kontrol bağımlılıkları ve veri bağımlılıklarının birleşimiyle oluşan bir yapıdır. Komondoor tarafından hazırlanan bir çalışmada programların bazı kesitleri kullanılarak programların bağımlılık grafiklerinin yapılarını bulmak amaçlanmıştır. Sheneamer ise yaptığı çalışmayla AST ve PDG kullanarak özelliklerin belirlenmesini sağlamış böylece fonksiyonel olarak kod klon tespiti yapmayı amaçlamıştır [34].

Bunların yanı sıra kod klon tespitlerinde bazı hibrit yaklaşımlarda kullanılmıştır. DeepSim bu tür yaklaşımlardan bir tanesi olarak karşımıza çıkmaktadır. Deepsim hem kod kontrol akış diyagramlarını kullanarak hem de veri akışını semantik bir matrise dönüştürür ve fonksiyonel kod klon tespiti için bir derin öğrenme yöntemi kullanır. DeepSim derin öğrenme tekniği kullanan bir kod klon tespit modelidir. Java programlama dili üzerinde yöntem seviyesinde uygulamalar yapılmıştır [10]. CCLearner da derin öğrenme tekniği kullanarak kod klon tespiti yapan diğer bir kod klon tespit araçlarındandır. SCDetector (Semantic Clone Detector) üç temel fazdan oluşan bir derin öğrenme kod klon tespit çalışmasıdır. Şekil 3.1'de SCDetector program kodlayıcı aşamaları verilmiştir. 1nci fazda statik analiz yapılır, 2nci fazda merkezi bir analiz yapılır, üçüncü fazda ise kod klon tespiti yapılır. Derin öğrenme

tekniklerinde en çok kullanan modellerden bir tanesi GRU (Gated Recurrent Unit) diğeri ise LSTM (Long Short-Term Memory) olarak karşımıza çıkmaktadır. SCDetector ise GRU modelini kullanmıştır ve sonrasında Siamese derin öğrenme algoritması ile kod klon tespiti yapmaktadır. Şekil 3.2'de SCDetector Siamese altyapısı gösterilmiştir. RtvNN, RNN-tabanlı bir kod klon tepit aracıdır [22]. ASTNN, ASTler üzerinde GRU modelini kullanan bir derin öğrenme destekli fonksiyonel kod klon tespit aracıdır [11].



Şekil 3.1 SCDetector program kodlayıcı aşamaları



Şekil 3.2 SCDetector Siamese altyapısı

2020 yılında yayınlanan bir diğer araştırma da sözdizimsel ve anlamsal kavramlar üzerinden fonksiyonel kod klon tespiti üzerine kurgulanmıştır. C++ dili üzerinde yazılmış projelere odaklanan bu çalışmada farklı teknikler denenmiştir. Derin öğrenme algoritmaları ile de desteklenen bu çalışmada AST, PDG ve CFG'den de

faydalanılmıştır. AST ve CFG hibrit bir şekilde kullanılarak sözdizimsel ve anlamsal olarak özellikler ortaya konmuştur. Bu özellikler üzerinden benzerlik analizleri ve hesaplamaları yapılmıştır. OJClone veri seti üzerinde 104 programlama görevi bulunan ve her görev için farklı yazılım geliştiriciler tarafından oluşturulmuş 500 kaynak kod ile deneyler yapılmıştır. Temelde üç adım ile kod klon tespitine katkı sağlanmıştır öncelikle modelin oluşturulması çalışmaları yapılmıştır daha sonra sözdizimsel ve anlamsal olarak modelleme yapılmış ve son olarak da derin öğrenme teknikleri ile kod klon tespitleri yapılmıştır. Modelleme yapılırken oluşturulan vektörler Graph2vec yöntemi ile oluşturulmuştur. Ayrıca çalışma sonuçlarında diğer kod klon tespit araçları ile olan kıyaslamalara da yer verilmiştir.

Tablo 3.1 Kod klon tespiti üzerine bilimsel araştırmaların karşılaştırmaları

Makale	Yıl	ISBN / ISSN	Platform	Yöntem	Veri Seti	Sonuçlar
Cross-language clone detection by learning over abstract syntax trees	2019	978-1-7281-3412-3	Çok dilli	Hibrit,	45,000 kaynak kod parçası içermektedir.	Model, F1 Skor, Hassasiyet(Precision) ve Tekrar çağrım (Recall) sonuçarı sırasıyla şöyledir; Temel model, 0.53 ,0.41, 0.74 Önceden eğitilmiş jeton vektörleri (veri içermez), 0.51, 0.40, 0.71 Rastgele başlatılan jeton vektörleri, 0.61, 0.49, 0.82 Önceden eğitilmiş jeton vektörleri, 0.66 , 0.55, 0.83
Deep learning code fragments for code clone detection	2016	978-1-4503-3845-5	Java	Ngram, Yinelemeli sinir ağları	ANTLR 4, Apache Ant 1.9.6, ArgoUML 0.34, CAROL 2.0.5, dnsjava 2.0.0, Hibernate 2x, JDK 1.4.2, JHotDraw 6	Dosya tabanlı AST tekniği için Sistem ve Hassasiyet değerleri şöyledir; ANTLR: 97% Apache Ant: 92% ArgoUML: 90% CAROL: 100% dnsjava: 47% Hibernate: 100% JDK: 90% JHotDraw: 100%
DeepSim Deep Learning Code Functional Similarity	2018	978-1-4503-5573-5	Java	Derin sinir ağları	The Google Code Jam	GCJ veriseti ile elde edilen sonuçlar şöyledir veriler sırasıyla yöntem, Geri Çağrım, Hassasiyet ve F1 Skoru olarak verilmiştir. DECKARD, 0.44, 0.45, 0.44 SdA-base, 0.51, 0.50, 0.50 SdA-unsup, 0.56, 0.26, 0.35 RtvNN, 0.90, 0.20, 0.33 DeepSim, 0.82, 0.71, 0.76 BigCloneBench veri seti ile elde edilen sonuçlar şöyledir veriler sırasıyla yöntem, Geri Çağrım, Hassasiyet ve F1 Skoru olarak verilmiştir. DECKARD, 0.02, 0.93, 0.03 RtvNN, 0.01, 0.95, 0.01 CDLH, 0.74, 0.92, 0.82 DeepSim, 0.98, 0.97, 0.98
BinDeep A deep learning approach to binary code similarity detection	2021	0957-4174	Çok dilli	Hibrit,	47 milyon fonksiyon ikilisi sayısı. 6 popüler Linux paketi.	LSTM, CNN ve CLSTM modellerinin sırasıyla Hassasiyet, Geri Çağrım ve F1 Skorları şöyledir; LSTM : 0.9179, 0.8812, 0.8991 CNN: 0.9149, 08679, 0.8907 CLSTM: 0.9860, 0.9965, 0.9769
Functional Code Clone Detection with Syntax and Semantics	2020	978-1-4503-8008-9	C/C++	Word2vec, AST, CFG	OJClone.	OJClone veriseti ile yapılan sonuçlar sırasıyla şöyledir; model, Hassasiyet, Geri Çağrım, F1 Skor Deckard, 0.99, 0.05, 0.10 SourcererCC, 0.07, 0.74, 0.14 DLC, 0.71, 0.00, 0.00 CDLH, 0.47, 0.73, 0.57 DeepSim, 0.70, 0.83, 0.76 ASTNN, 0.98, 0.92, 0.95 Araştırmada uygulanan model, 0.97, 0.95, 0.96
Detecting Code Clones with Graph Neural Network and Flow-Augmented Abstract Syntax Tree	2020	978-1-7281-5143-4	Java	Grafik gömme modeli, Grafik eşleştirme ağları	Google Code Jami BigCloneBench	GCJ veri seti ile elde edilen sonuçlar için sırasıyla Model, Hassasiyet, Geri Çağrım ve F1 Skorları şöyledir; Deckard, 0.45, 0.44, 0.44 RtvNN, 0.20, 0.90, 0.33 ASTNN, 0.98, 0.93, 0.95 FA-AST+GGNN, 0.96, 1.0, 0.97 FA-AST+GMN, 0.99, 0.97, 0.98
CCEyes An Effective Tool for Code Clone Detection on Large-Scale Open Source Repositories	2021	978-1-6654-1919-2	Java	Kösinüs benzerliği	BigCloneBench	Sırasıyla Model, Geri Çağrım, Hassasiyet ve F1 Skorları şöyledir; Weighted RAE, 0.55, 0.98, 0.71 CDLH, 0.74, 0.92, 0.82 Deepsim, 0.98, 0.97, 0.975 CCEyes, 0.965, 0.999, 0.982

CLCDSA Cross Language Code Clone Detection using Syntactical Features and API Documentation	2019	978-1-7281-2508-4	Çok dilli	Kösinüs benzerliği	Java, C# ve Python ile oluşturulmuş 78000'den fazla kod içeriği.	Sırasıyla Model, Hassasiyet, Geri Çağrım ve F1 Skorları şöyledir; LICCA, 0.14, 0.32, 0.194 CLCMiner, 0.09, 0.13, 0.11 AST Learner, 0.184, 0.81, 0.30
Fast Code Clone Detection Based on Weighted Recursive Autoencoders	2019	2169-3536	Java	Hızlı yaklaşık en yakın komşu arama Fast ANNS (Aproximate nearist neighbor Search)	BigCloneBench	Sırasıyla Model, Geri Çağrım ve Hassasiyet sonuçları şöyledir; Oreo, 0.89, 0.895 CCLearner, 0.89, 0.93 SourceCC, 0.60, 0.978 CloneWorks, 0.93, 0.987 Nicad, 0.93, 0.99 Deckard, 0.31, 0.348 Araştırmada kullanılan yönem, 0.366, 0.9962
FCCA Hybrid Code Representation for Functional Clone Detection Using Attention Networks	2020	1558-1721	Java	Derin tekrarlayan sinir ağları	6351 derlenebilir Java dosyası, 275777 yöntem seviyesinde kod klon ikilisi. 269032 klon olmayan ikili sayısı.	SırasıylaModel, Hassasiyet, Geri Çağrım ve F1 Skorları şöyledir; DECKARD, 0.93, 0.002, 0.003 DLC, 0.95, 0.01, 0.01 SOURCERERCC, 0.88, 0.02, 0.03 CDLH(AST), 0.92, 0.74, 0.82 TBCNN, 0.90, 0.81, 0.85 DEEPSIM, 0.97, 0.98, 0.98 FOCA, 0.98, 0.97, 0.98
Neural detection of semantic code clones via tree-based convolution	2019	978-1-7281-1519-1	Çok dilli	Kösinüs benzerliği	BigCloneBench, OJClone	
Oreo detection of clones in the twilight zone	2018	978-1-4503-5573-5	Java	Derin Sinir Ağları	Açık kaynak kodlu GitHub projeleri. 25Milyon vektör.	BigCloneBench veri seti ile oluşturulan sonuçlar için sırasıyla Model, Hassasiyet, Geri Çağrım ve F1 Skorları şöyledir; Deckard, 0.93, 0.02, 0.03 DLC, 0.95, 0.01, 0.01 SourcerereCC, 0.88, 0.02, 0.03 CDLH, 0.92, 0.74, 0.82 TBCCD 0.97, 0.96, 0.97 OJClone veri seti ile oluşturulan sonuçlar için sırasıyla Model, Hassasiyet, Geri Çağrım ve F1 Skorları şöyledir; Deckard, 0.99, 0.05, 0.10 DLC, 0.71, 0.00, 0.00 SourcerereCC, 0.07 0.74, 0.14 CDLH, 0.47, 0.73, 0.57 TBCCD 0.99, 0.99, 0.99
SCDetector software functional clone detection based on semantic tokens analysis	2021	978-1-4503-6768-4	Çok dilli	CFG	Google Code Jam, BigCloneBench	BigCloneBench veri seti ile oluşturulan sonuçlar için sırasıyla Model, Geri Çağrım, Hassasiyet ve F1 Skorları şöyledir; SourcererCC, 0.07, 0.98, 0.14 Deckard, 0.06, 0.93, 0.12 RtvNN, 0.01, 0.95, 0.01 ASTNN, 0.94, 0.92, 0.93 SCDetector, 0.97, 0.98, 0.98 Google Code Jam veri seti ile oluşturulan sonuçlar için sırasıyla Model, Geri Çağrım, Hassasiyet ve F1 Skorları şöyledir; SourcererCC, 0.11, 0.43, 0.17 Deckard, 0.44, 0.45, 0.44 RtvNN, 0.90, 0.20, 0.33 SCDetector, 0.87, 0.81, 0.82

4. YÖNTEM

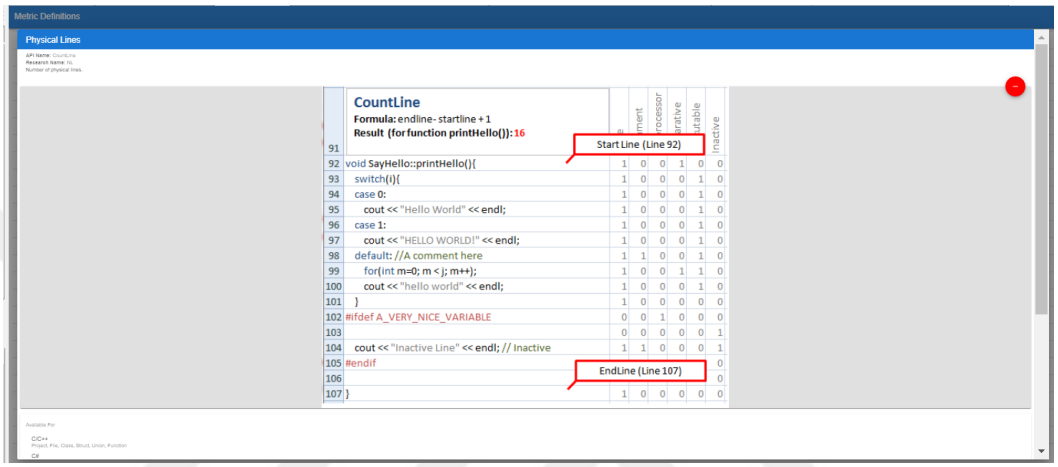
Deneylerimiz platform içi ve platformlar arası olacak şekilde hem aynı programlama dillerinde kod klon tespitleri hem de farklı programlama dilleri arasında kod klon tespitlerini ortaya koymak ve katkıda bulunmak üzere geliştirilmiştir. Literatürde platformlar arası kod klon tespitleri üzerine çok fazla araştırma olmadığı gözlemlenmiştir. Bilimsel araştırmalar ve yapılan çalışmalar incelenmiştir. Kod klon tespit araçları araştırılıp detaylı bir şekilde yapıları incelenmiştir. Kod klon türleri üzerine araştırmalar yapılmıştır. Kod klonlarını belirlemede kullanılan teknikler araştırılmıştır. Bu tekniklerin başarı oranları ve çeşitlilikleri incelenmiştir. Tekniklerde kullanılan yöntemler ve metrikler araştırılmıştır ve sonuç olarak deneylerimizde kullandığımız yöntemler kararlaştırılmıştır. Deney çıktıları detaylı olarak sonuç bölümünde paylaşılmıştır.

Deneylerimiz dünya üzerinde popüler olan ve kullanım oranları ile listelerde ilk sıralarda yer alan dört programlama dili üzerinden ilerletilmiştir. Java, C, C# ve JavaScript programlama dilleri üzerinde geliştirilmiş veri kümeleri hazırlanmıştır. Hazırlanan veri kümeleri incelenerek benzerliklerini bulabilmek adına metrikler araştırılmıştır. Araştırmalar sonucunda 104 farklı metrik ortaya çıkartılmıştır. Hem veri kümeleri hem de programlama dillerinin kullanım alanları ve kısıtlarından dolayı ortak olarak Tablo 4.1'de verilen 34 farklı metrik kullanılmıştır.

Tablo 4.1 Understand Uygulamasında Kullanılan Metrikler

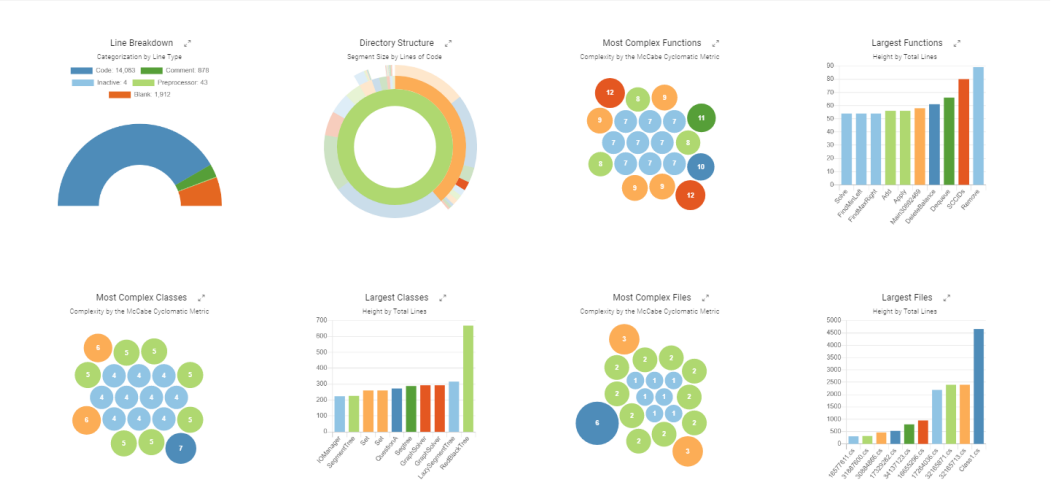
Metrik	Kısaltma	Açıklama
Average Cyclomatic Complexity	AvgCyclomatic	Tüm iç içe işlevler veya yöntemler için ortalama döngüsel karmaşıklık.
Average Modified Cyclomatic Complexity	AvgCyclomaticModified	Tüm iç içe işlevler veya yöntemler için ortalama değiştirilmiş döngüsel karmaşıklık.
Average Strict Cyclomatic Complexity	AvgCyclomaticStrict	Tüm iç içe işlevler veya yöntemler için ortalama katı döngüsel karmaşıklık.
Average Essential Cyclomatic Complexity	AvgEssential	Tüm iç içe işlevler veya yöntemler için ortalama temel karmaşıklık.
Average Number of Lines	AvgLine	Tüm iç içe işlevler veya yöntemler için ortalama satır sayısı.
Average Number of Blank Lines	AvgLineBlank	Tüm iç içe işlevler veya yöntemler için ortalama boşluk sayısı.
Average Number of Lines of Code	AvgLineCode	Tüm iç içe işlevler veya yöntemler için kaynak kodu içeren satırların ortalama sayısı.
Average Number of Lines with Comments	AvgLineComment	Tüm iç içe işlevler veya yöntemler için yorum içeren satırların ortalama sayısı.
Base Classes	CountClassBase	Acil temel sınıfların sayısı.
Classes	CountDeclClass	Sınıf sayısı.
Executable Unit	CountDeclExecutableUnit	Yürütülebilir koda sahip program birimlerinin sayısı.
Function	CountDeclFunction	İşlev sayısı.
Physical Lines	CountLine	Tüm satırların sayısı.
Blank Lines of Code	CountLineBlank	Boş satır sayısı.
Source Lines of Code	CountLineCode	Kaynak kodu içeren satır sayısı.
Declarative Lines of Code	CountLineCodeDecl	Bildirime dayalı kaynak kodu içeren satır sayısı.
Executable Lines of Code	CountLineCodeExe	Yürütülebilir kaynak kodunu içeren satır sayısı.
Lines with Comments	CountLineComment	Yorum içeren satır sayısı.
Inactive Lines	CountLineInactive	Etkin olmayan hatların sayısı.
Preprocessor Lines	CountLinePreprocessor	Önişlemci satır sayısı.
Semicolons	CountSemicolon	Noktalı virgül sayısı.
Statements	CountStmt	İfade sayısı.
Declarative Statements	CountStmtDecl	Bildirim beyanlarının sayısı.
Executable Statements	CountStmtExe	Yürütülebilir deyimlerin sayısı.
Max Cyclomatic Complexity	MaxCyclomatic	İç içe geçmiş tüm işlevlerin veya yöntemlerin maksimum döngüsel karmaşıklığı.
Max Modified Cyclomatic Complexity	MaxCyclomaticModified	Yuvalanmış işlevlerin veya yöntemlerin maksimum değiştirilmiş döngüsel karmaşıklığı.
Max Strict Cyclomatic Complexity	MaxCyclomaticStrict	Yuvalanmış işlevlerin veya yöntemlerin maksimum katı döngüsel karmaşıklığı.
Max Essential Complexity	MaxEssential	Tüm iç içe işlevlerin veya yöntemlerin maksimum temel karmaşıklığı.
Nesting	MaxNesting	Kontrol yapılarının maksimum iç içe geçme düzeyi.
Comment to Code Ratio	RatioCommentToCode	Yorum satırlarının kod satırlarına oranı.
Sum Cyclomatic Complexity	SumCyclomatic	İç içe geçmiş tüm işlevlerin veya yöntemlerin döngüsel karmaşıklığının toplamı.
Sum Modified Cyclomatic Complexity	SumCyclomaticModified	İç içe geçmiş tüm işlevlerin veya yöntemlerin değiştirilmiş döngüsel karmaşıklığının toplamı.
Sum Strict Cyclomatic Complexity	SumCyclomaticStrict	İç içe geçmiş tüm işlevlerin veya yöntemlerin kesin döngüsel karmaşıklığının toplamı.
Sum Essential Complexity	SumEssential	İç içe geçmiş tüm işlevlerin veya yöntemlerin temel karmaşıklığının toplamı.

Bu metriklerin hemen hemen hepsini sağladığı için “Understand” [35] uygulaması kullanılmıştır. Bu metrikler uygulama içerisinde detaylı bir şekilde anlatılmakta ve hangi amaç için kullanıldığı örnek veriler ile sunulmaktadır. Örnek bir metrik detayı Şekil 4.1’de verilmiştir. "Understand", platformlar arası çalışabilen, çoklu dil desteği olan, bakım odaklı bir IDE'dir (Etkileşimli Geliştirme Ortamı). Büyük çaplı eski veya yeni kaynak kodlu projelerin bakım çalışmalarının yapılmasına yardımcı olmakla birlikte kod analizi yapmaya yardımcı olmaktadır.



Şekil 4.1 “CountLine” Metriğinin Açıklaması

Ayrıca farklı görselleştirmeler ile kaynak kodlara büyük çerçeveden bakabilme imkânı sunmaktadır. Şekil 4.2’de örnek bir projenin özet görüntüsünün grafikler halinde görselleştirilmesi verilmiştir.



Şekil 4.2 Understand uygulaması üzerinde örnek bir proje ile ilgili özet görselleştirme

Deneylerimize veri oluştururken bir yazılım yarışma içeriği sunan AtCoder [36] uygulamasından faydalandık. AtCoder, yazılıma yeni başlamış veya uzman her

seviyede yazılım geliştiricinin yarışmaya katılabileceği Japonya merkezli bir yazılım geliştirme yarışma programıdır. Haftalık çevrimiçi yarışmalar yaparak yazılım geliştiricilere kendilerini geliştirebilme imkânı sunmaktadır. AtCoder'da üç kategoride resmi yarışma yapılmaktadır. AtCoder Grand Contest (AGC). Bu AtCoder'ın en iyi yarışmasıdır. Problemler yüksek özgünlüğe sahiptir ve ilginç gözlemler gerektirmektedir. AtCoder Regular Contest (ARC). AGC problemlerine kıyasla problemler biraz tipik olabilir, ancak yine de yazılım geliştiricilerin çoğunun ilgisini çekebilen ve pratik yapabileceği problemlerdir. AtCoder Beginner Contest (ABC). Bu kategori, esas olarak rekabetçi programlamaya yeni olanlar için hedeflenmiştir. Sorular kolay ve öğreticidir. AtCoder'ın bir derecelendirme sistemi bulunmaktadır. AtCoder yarışmaların sonuçlarını web sayfası üzerinden yayınlamaktadır. Biz araştırmalarımızı bu web sayfası üzerinden belirlediğimiz 4 problem üzerinden çözümü onaylanmış kayıtlar ile bir veritabanı oluşturarak gerçekleştirdik.

Tüm deneylerde ve testlerde kullandığımız dört farklı senaryo şu şekilde başlıklandırılmıştır. Detayları EK A.1, EK A.2, EK A.3 ve EK A.4'te verilmiştir.

Senaryo1: Children and Candies [37]

Senaryo2: Iroha Loves Strings [38]

Senaryo3: Tak and Cards [39]

Senaryo4: Many Formulas [40]

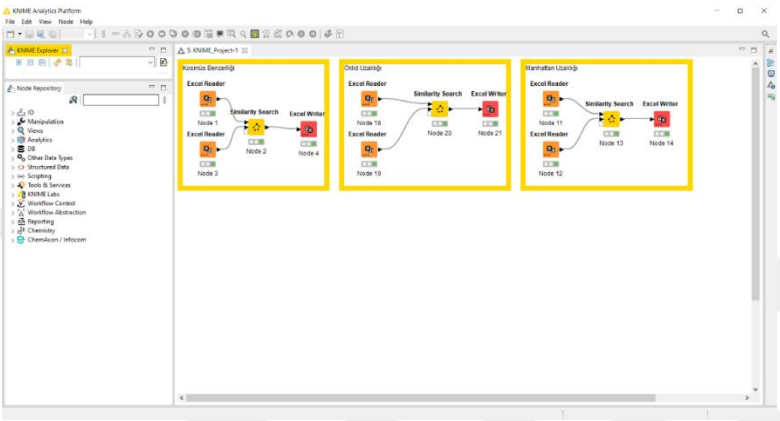
Bu senaryolar için topladığımız veriler ile ilgili platform, çözüm dosya sayıları, ilgili çözümlerin kod satır sayıları ve çözümlerdeki yorum satır sayıları Tablo 4.2'de sunulmuştur.

Tablo 4.2 Programlama dili, senaryo dosyası ve metrik sayıları.

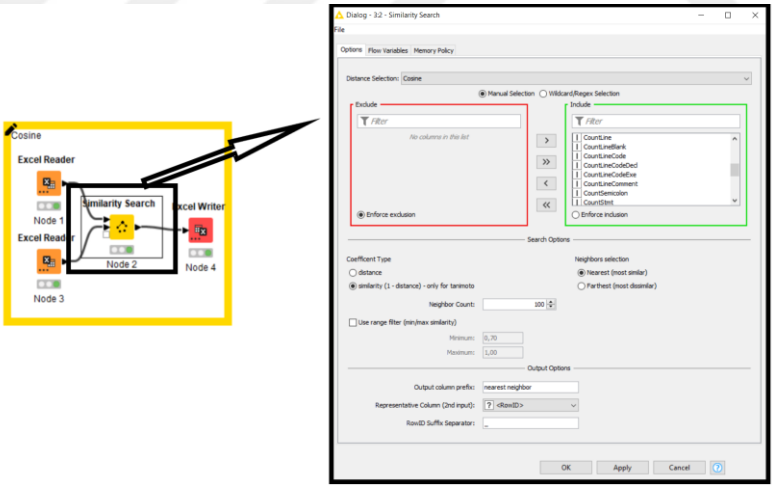
Platform	**Kaynak Kod Çözüm Sayısı**	**Kod Satır Sayısı**	**Kod Yorum Satır Sayısı**
C#	315	41045	2328
C	80	1616	77
JavaScript	65	1249	112
Java	160	7815	475

Oluşturulan verilerin analizlerini yapmak için KNIME Analytics Platformu'ndan faydalanılmıştır. KNIME Analytics Platformu veri bilimi oluşturmaya yönelik

ücretsiz, açık kaynaklı bir yazılımdır. KNIME Analytics modern yazılım mühendisliği standartları uygulanan, çeşitli veri yükleme, dönüştürme, analiz ve görselleştirme modüllerini kapsayan kaliteli, modüler ve yüksek düzeyde ölçeklenebilir bir platformdur. Benzerlik ölçümlerini yaparken kullandığımız modüllerinde yaptığımız akışlar Şekil 4.3'te verilmiştir. Ölçümler sırasında kullanılan verilerin parametreleri ve ölçüm ayarlamaları yapılabilmektedir, örnek bir gösterim de Şekil 4.4'te gösterilmiştir.



Şekil 4.3 Knime Analytics Platformu üzerinde benzerlik ölçümlerinin yapılması



Şekil 4.4 Knime Analytics Platformu üzerinde benzerlik değerlerinin bulunmasında kullanılan metrikler ve kriterler ekranı

KNIME Analytics birçok veri biliminde analiz yapabilme kabiliyetine sahiptir. Ayrıca verilerin görselleştirilmesi noktasında da gelişmiş özellikler sunmaktadır. Bu araştırmada KNIME Analytics uygulamasından veri yükleme, veri analiz etme, verilerin sınıflandırılması, temizlenmesi, işlenmesi ve hesaplamaların yapılması noktalarında faydalanılmıştır. Bu araştırmamızda kullandığımız senaryolar için veriler

toplandıktan sonra bu veriler tek tek işlenerek metriklerin oluşturulduğu vektörel kümeler ortaya çıkartılmıştır. Araştırmada kullanılan tüm programlama dillerini kapsayacak ortak metrikler üzerinden ilerlenmiştir. Metriklerdeki araştırmaya katkısı olmayan ve ayırt edici bir bilgi sunmayan veriler temizlenmiştir. Daha sonra üç farklı teknik ile hesaplamalar yapılmıştır. Hesaplamalarda Kosinüs benzerliği, Öklid uzaklığı ve Manhattan uzaklığı teknikleri kullanılmıştır.

Kosinüs benzerliği metinler arasındaki benzerliği vektörel olarak ölçmeye yaran bir hesaplama tekniğidir. Kosinüs benzerliği iki vektör arasındaki benzerliğin Kosinüs açısı hesaplanarak bulunmaktadır. A ve B vektörlerinin Kosinüs benzerliği denklem (4.1)'de verildiği gibi hesaplanmaktadır. Kosinüs benzerliğinin değer aralığı 0 ve 1 arasında gerçekleşmektedir. Hesaplamalar sonucu bulunan değer 1'e ne kadar yakınsa iki vektör birbirine o kadar benzerdir. Kosinüs benzerlik değeri 0 ise iki vektör birbiri ile alakasız olarak değerlendirilmektedir. Tüm vektör elemanları aynı değere sahip ve birebir olduğu durumda Kosinüs benzerlik değeri 1 olarak hesaplanmaktadır.

A vektörü şu şekilde verilmiştir; $A = [a_1, a_2, \dots, a_n]$

B vektörü şu şekilde verilmiştir; $B = [b_1, b_2, \dots, b_n]$

A ve B vektörleri arasındaki açı θ açısı olarak verilmiştir.

A ve B vektörleri arasındaki θ açısının kosinüs değeri($\cos(\theta)$) denklem (4.1)'de verilen formül ile hesaplanabilmektedir.

$$\cos(\theta) = \frac{A \cdot B}{||A||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{4.1}$$

Denklem (4.1) ile verilen formül ile kod klon tespitinde kullanacağımız kosinüs benzerliği hesaplama formülü verilmiştir. Kosinüs açısı ile hesaplanan bu değer iki vektörün birbirine olan ilişkisinin bir açı olarak ifade edilmesidir. Örneğin tamamen aynı yönü gösteren yani birebir aynı olan iki vektörün kosinüs açı değeri 1 olarak hesaplanacaktır. İki vektör arasında bulunan açının kosinüs değeri A ve B vektörlerinin

nokta çarpımının, vektörlerin uzunluklarının çarpımına oranı ile hesaplanmaktadır. Nokta çarpımı ile ilgili formül denklem (4.2)'de verilmiştir.

$$A \cdot B = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n$$

(4.2)

A vektörünün uzunluğu hesaplanırken A vektöründe bulunan her elemanın karesinin kare kökü alınarak denklem (4.3)'de verildiği gibi hesaplanmıştır.

$$|\mathrm{A}| = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}$$

(4.3)

B vektörünün uzunluğu hesaplanırken B vektöründe bulunan her elemanın karesinin kare kökü alınarak denklem (4.4)'te verildiği gibi hesaplanmıştır. Bulunan değerler çarpılarak kosinüs değerini bulduğumuz formülün paydası elde edilmiştir.

$$|B| = \sqrt{b_1^2 + b_2^2 + \cdots + b_n^2}$$

(4.4)

Öklid uzaklığı, iki nokta arasındaki doğrusal uzaklık olarak tanımlanmaktadır. Tek boyutta hesaplanabildiği gibi çok boyutlu kartezyen veri uzayında da iki nokta arasındaki uzaklığın doğrusal bağlantı yöntemi ile de ölçülebilir. n boyutlu Öklid uzayında P kümesi şu şekilde verilmiştir; P =(p_1, p_2, ..., p_n). Q kümesi ise şu şekilde verilmiştir; Q = (q_1,q_2, … ,q_n). P ve Q kümeleri arasındaki Öklid uzaklığının hesaplanması Denklem (4.5)'te verildiği gibi formülleştirilmiştir.

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

(4.5)

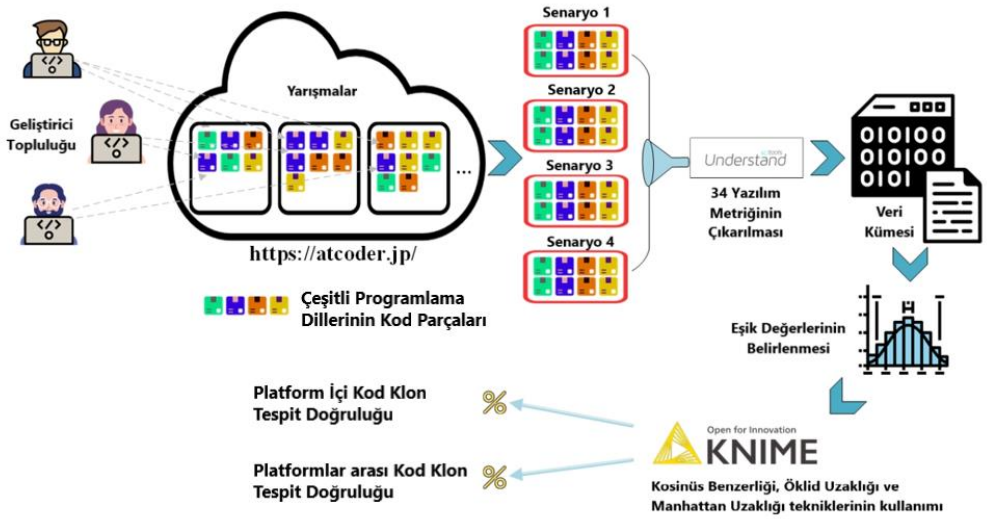
p ve q iki nokta olarak verildiğinde bu iki nokta arasındaki doğrusal uzaklık Öklid ölçümünden faydalanarak formül (4.6)'da verildiği hesaplanmaktadır.

$$d(p,q) = |\mathrm{p} - \mathrm{q}| \tag{4.6}$$

Manhattan uzaklığı, iki nokta arasındaki mesafenin ölçülmesi için kullanılmaktadır. İki nokta arasındaki mesafenin kartezyen koordinatlarının mutlak farklarının toplamı ile hesaplanmaktadır. x ve y ikililerine sahip d kümesi için Manhattan uzaklığının ölçümü denklem (4.7)'de formülleştirilmiştir.

$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i| \tag{4.7}$$

Bu üç tekniğin kod benzerlikleri bulma yönünde kullanılması ve kod benzerlikleri bulunmasındaki başarı oranlarını ortaya çıkartacak çalışmalar yapılmıştır. Tüm tekniklerde kaynak kodlardan elde ettiğimiz veriler ile oluşturulan metrikler kullanılmıştır. Metriklerin değerleri ile vektörel kümeler oluşturulmuştur. Formüllerde verilen uzaklık ve benzerlik hesaplamalarında bu kümelerden faydalanılmıştır. Platform içi ve platformlar arası kod klon tespiti senaryolarımızda 34 farklı metrikten oluşan veriler vektörel olarak ikililer halinde sırasıyla Kosinüs benzerliği, Öklid uzaklığı ve Manhattan uzaklığı formülleri ile tek tek hesaplanmış ve benzerlik değerleri bulunmuştur. Çalışmadan elde edilen bulgular beşinci bölümde detaylı olarak verilmiştir.



Şekil 4.5 Kod Klonlarını Tanımlamak İçin Önerilen Yöntemdeki Adımlar

Kod klonlarını bulmak için tercih ettiğimiz yaklaşım, sıralı adımlardan oluşan bir zincirdir. AtCoder portalının kod parçacıkları, Understand uygulaması kullanılarak yazılım ölçümleri çıkarılarak bir veri kümesine dönüştürülür. Üretilen veri kümesi, belirli bir senaryoya cevap olarak açıklamalı kod parçacıkları içerir ve bunların ne kadar benzer olduklarını belirlemede çeşitli yaklaşımların uygulanmasına izin verir. Şekil 4.5 bu prosedür boyunca meydana gelen olayların sırasını göstermektedir.

5. BULGULAR

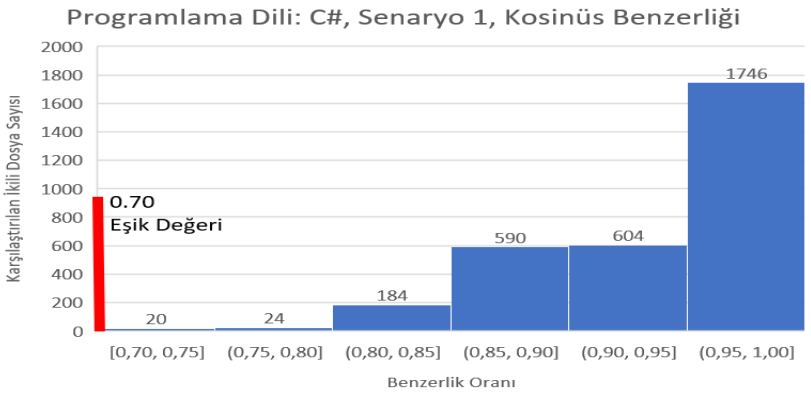
Bu bölümde, yapılan analiz ve test çalışmaları ile elde edilen bulgular ve bu bulgular ile yorumlara dayalı olarak elde edilen sonuçlar yer almaktadır. Deneylerimizde kullanılmak üzere C, C#, Java ve JavaScript programlama dili kullanılmıştır. Kod klonlarının tespiti aşamasında kullanılan teknikler olarak; Kosinüs Benzerliği, Öklid Uzaklığı ve Manhattan Uzaklığı belirlenmiştir.

Dört farklı senaryo için dört farklı programlama dilinde oluşturulan kaynak kodları derlenmiştir. Toplamda 620 dosya toplanmış ve her programlama dili önce kendi içerisinde daha sonra da çapraz olacak şekilde diğer programlama dilleri ile kod klon tespiti üzerine deneyler yapılmıştır. Aynı platformdaki benzer çözümler için üretilen kodlar için kıyaslamalar yapılmıştır. Farklı platformdaki benzer çözümler için üretilen kodlar için kıyaslamalar yapılmıştır. Tek tek ve toplu olarak karışık ölçümlemeler yapılmış ve bu ölçümlemelerdeki benzerlik oranlarını belirlerken seçilen kriterlerin oluşumunda veriye dayalı bir sınıflandırma yapılmıştır. Tüm dosyaların ölçümlerinden oluşan veriler sınıflandırılarak ortaya histogram grafikleri çıkartılmış ve kritik eşik değerleri bu verilere dayalı seçilmiştir. Orta Nokta Yöntemi kullanılarak, verilerin en az %90'ını oluşturan eşik değerleri bulunmuştur. Uygun bir eşik konumu belirlemek için, yinelemeli olarak Orta Nokta Yöntemini kullandık. Bir eşik değeri belirlemek için ilk görünen değerlerin ortancasını aldık. Ardından, bu sınırın üstündeki ve altındaki değerler arasındaki yüzde dağılımını analiz ettik. Bu yöntemde kullandığımız yöntem formül 5.1'de verilmiştir.

$$\left(\frac{x_1 + x_2}{2}\right), \left(\frac{y_1 + y_2}{2}\right) \tag{5.1}$$

Alt sınır ve üst sınır, frekans tablosundaki her bir sınıf aralığının veya grubun alt ve üst sınırlarını ifade eder. Orta nokta tekniği kullanılırken, her bir aralığın orta nokta değerini belirlemek için frekans dağılımındaki veya histogramdaki her bir aralığa veya gruba bu formülün uygulanması gerekir. Her aralık için medyan sayıları alır almaz, seçilen benzerlik yöntemlerini uygularız. Benzerlik oranlarını belirlediğimiz eşik değerlerini bulurken analizler sonucunda ortaya çıkan verileri kullandık. Bu çalışmalar sonucunda Kosinüs benzerliği için yaptığımız eşik değeri belirleme

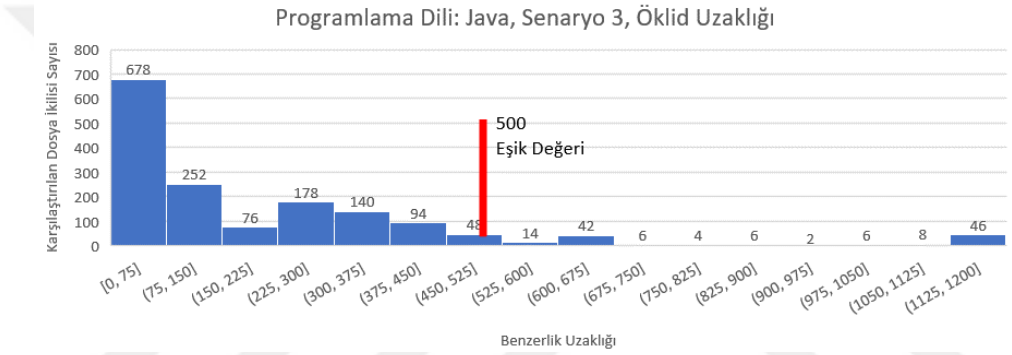
analizlerinden birisi olan C# programlama dili ile yapılmış ölçüm Şekil 5.1'de verilmiştir. "Eşik Değeri" olarak belirtilmiş olan değer tüm verilerdeki ölçümlerimiz sonucu ortaya çıkan kod klon tespiti yaparken kullanacağımız değerdir. Kosinüs benzerliği tekniğinde 0.70 değeri Eşik Değer olarak belirlenmiş olup bu değer altında kalan ölçüm sonuçları benzer olmayan kodlar olarak değerlendirilmiştir. Şekil 5.1'de verilen verilerden de görüldüğü gibi Senaryo1 (Children and Candies) için 0,70 değeri üzerinde bulunan ölçümler benzer kodlar olarak değerlendirilmiştir. Şekil 5.1'de dikey eksende bulunan 0 ile 2000 aralığı benzerliği kıyaslanan ikili dosyaların sayısını temsil etmektedir. Yatay eksende verilen benzerlik oranları ise benzerliği kıyaslanan ikili dosyaların ne kadar benzer olup olmadığını belirten oransal değeri vermektedir. Benzerlik oranları 1'e yaklaştıkça kodların özdeş olma özellikleri yükselmektedir. 58 dosyanın kartezyen çarpımları sonucu ortaya çıkan ikili karşılaştırmalardan elde ettiğimiz verilere göre 1746 dosya ikilisi %95 ve %100 oranında benzerdir. 604 dosya ikilisi %90 - %95 aralığında benzerdir. 590 dosya ikilisi %85 - %90 oranında benzerdir. 184 dosya ikilisi %80 - %85 oranında benzerdir. 24 dosya ikilisi %75 - %80 oranında benzerdir. 20 dosya ikilisi de %70 - %75 oranında benzerdir.



Şekil 5.1 Senaryo1 Kosinüs Benzerliği Histogram Grafiği – C#

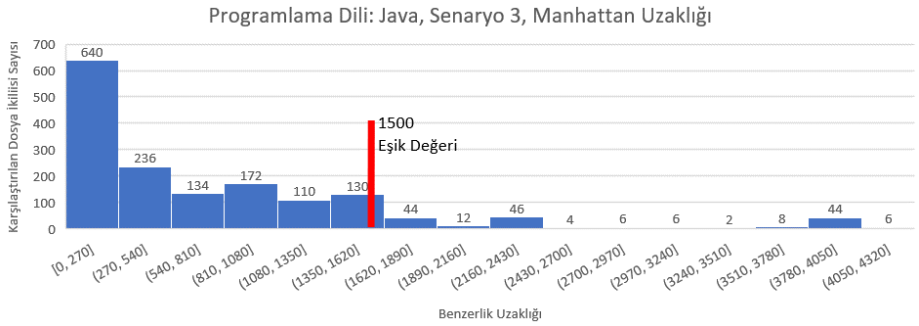
Öklid uzaklığı tekniğinde 500 değeri Eşik Değer olarak belirlenmiş olup bu değer altında kalan ölçüm sonuçları benzer olmayan kodlar olarak değerlendirilmiştir. Şekil 5.2'de verilen verilerden de görüldüğü gibi Senaryo3 (Tak and Cards) için 500 değerinin altında bulunan ölçümler benzer kodlar olarak değerlendirilmiştir. Şekil 5.2'de dikey eksende verilen 0 ile 800 arasındaki değer aralığı benzerliği kıyaslanan ikili dosyaların sayısını temsil etmektedir. Yatay eksende verilen uzaklık değeri ise benzerliği kıyaslanan ikili dosyaların ne kadar benzer olup olmadığını belirten değeri göstermektedir. Benzerlik, uzaklık değeri 0'a yaklaştıkça kodların özdeş olma

özellikleri yükselmektedir. 40 dosyanın kartezyen çarpımları sonucu ortaya çıkan ikili karşılaştırmalardan elde ettiğimiz verilere göre 678 dosya ikilisi 0-75 arası uzaklıkta olacak şekilde benzerlik göstermiştir. 252 dosya ikilisi 75 – 150 arası uzaklıkta olacak şekilde benzerlik göstermiştir. 76 dosya ikilisi 150 – 225 arası uzaklıkta olacak şekilde benzerlik göstermiştir. 178 dosya ikilisi 225 – 300 arası uzaklıkta olacak şekilde benzerlik göstermiştir. 140 dosya ikilisi 300 – 375 arası uzaklıkta olacak şekilde benzerlik göstermiştir. 94 dosya ikilisi 375 – 450 arası uzaklıkta olacak şekilde benzerlik göstermiştir. 46 dosya ikilisi 450 – 500 arası uzaklıkta olacak şekilde benzerlik göstermiştir. 500 değerinin üzerinde kalan dosya ikilileri benzer kodlar olarak değerlendirilmemiştir.



Şekil 5.2 Senaryo3 Öklid Uzaklığı Histogram Grafiği – Java

Manhattan uzaklığı tekniğinde 1500 değeri Eşik Değer olarak belirlenmiş olup bu değer altında kalan ölçüm sonuçları benzer olmayan kodlar olarak değerlendirilmiştir. Şekil 5.3'te verilen verilerden de görüldüğü gibi Senaryo3(Tak and Cards) için 1500 değeri altında bulunan ölçümler benzer kodlar olarak değerlendirilmiştir. Benzerlik oranları 0'a yaklaştıkça kodların özdeş olma özellikleri yükselmektedir. 40 dosyanın kartezyen çarpımları sonucu ortaya çıkan ikili karşılaştırmalardan elde ettiğimiz verilere göre 640 dosya ikilisi 0-270 arası uzaklıkta olacak şekilde benzerlik göstermiştir. 236 dosya ikilisi 270 – 540 arası uzaklıkta olacak şekilde benzerlik göstermiştir. 134 dosya ikilisi 540 – 810 arası uzaklıkta olacak şekilde benzerlik göstermiştir. 172 dosya ikilisi 810 – 1080 arası uzaklıkta olacak şekilde benzerlik göstermiştir. 110 dosya ikilisi 1080 – 1350 arası uzaklıkta olacak şekilde benzerlik göstermiştir. 120 dosya ikilisi 1350 – 1500 arası uzaklıkta olacak şekilde benzerlik göstermiştir. 1500 değerinin üzerinde kalan dosya ikilileri benzer kodlar olarak değerlendirilmemiştir.



Şekil 5.3 Senaryo3 Manhattan Uzaklığı Histogram Grafiği – Java

Dört programlama dili içinde önce kendi içinde daha sonra da kaynak kodları barındıran dosyalar karıştırılarak deneyler yapılmıştır. Tüm yapılan deneyler sonucunda dört senaryo ve dört programlama dili için analiz sonuçları verilmiştir. Tablo 5.1'den de görülebileceği gibi oluşturduğumuz senaryolar, kullanılan platform veya programlama dili, kullanılan kod benzerlik metodu, kaynak kodları barındıran dosyaların sayısı, toplam benzerlik ölçümü için karşılaştırılan dosya ikililerinin sayısı, analizler sonucu bulunan eşik değeri üzerinde ve altında bulunan kayıt sayıları ve sonuç olarak da kullanılan tekniğin kod klon benzerliğindeki üzerindeki başarı oranı gösterilmiştir.

Tablo 5.1 Programlama Dilleri ve Tekniklerin Analiz Sonuçları

Senaryo	Platform	Teknik	Dosya Sayısı	Toplam Karşılaştırılan Kayıt Sayısı	Eşik Değeri Üzerinde Bulunan Kayıt Sayısı	Eşik Değeri Altında Bulunan Kayıt Sayısı	Başarı Oranı (%)
Senaryo1	C#	Kosinüs Benzerliği	58	3364	3171	193	94,26
Senaryo2	C#	Kosinüs Benzerliği	54	2916	2591	325	88,85
Senaryo4	C#	Kosinüs Benzerliği	60	3600	3355	245	93,19
Senaryo3	C#	Kosinüs Benzerliği	54	2916	2880	36	98,77
Senaryo1	C#	Öklid Uzaklığı	58	3364	2770	594	82,34
Senaryo2	C#	Öklid Uzaklığı	54	2916	2550	366	87,45
Senaryo4	C#	Öklid Uzaklığı	60	3600	2482	1118	68,94
Senaryo3	C#	Öklid Uzaklığı	54	2916	2258	658	77,43
Senaryo1	C#	Manhattan Uzaklığı	58	3364	2754	610	81,87
Senaryo2	C#	Manhattan Uzaklığı	54	2916	2472	444	84,77
Senaryo4	C#	Manhattan Uzaklığı	60	3600	2412	1188	67,00
Senaryo3	C#	Manhattan Uzaklığı	54	2916	2206	710	75,65
Senaryo1	Java	Kosinüs Benzerliği	40	1600	1317	283	82,31
Senaryo2	Java	Kosinüs Benzerliği	40	1600	1586	14	99,13
Senaryo4	Java	Kosinüs Benzerliği	40	1600	1600	0	100,00
Senaryo3	Java	Kosinüs Benzerliği	40	1600	1582	18	98,88
Senaryo1	Java	Öklid Uzaklığı	40	1600	1532	68	95,75
Senaryo2	Java	Öklid Uzaklığı	40	1600	1522	78	95,13
Senaryo4	Java	Öklid Uzaklığı	40	1600	1600	0	100,00
Senaryo3	Java	Öklid Uzaklığı	40	1600	1462	138	91,38
Senaryo1	Java	Manhattan Uzaklığı	40	1600	1532	68	95,75
Senaryo2	Java	Manhattan Uzaklığı	40	1600	1522	78	95,13
Senaryo4	Java	Manhattan Uzaklığı	40	1600	1600	0	100,00
Senaryo3	Java	Manhattan Uzaklığı	40	1600	1376	224	86,00
Senaryo1	C	Kosinüs Benzerliği	20	400	390	10	97,50
Senaryo2	C	Kosinüs Benzerliği	20	400	360	40	90,00
Senaryo4	C	Kosinüs Benzerliği	20	400	372	28	93,00
Senaryo3	C	Kosinüs Benzerliği	20	400	380	20	95,00
Senaryo1	C	Öklid Uzaklığı	20	400	390	10	97,50
Senaryo2	C	Öklid Uzaklığı	20	400	360	40	90,00
Senaryo4	C	Öklid Uzaklığı	20	400	372	28	93,00
Senaryo3	C	Öklid Uzaklığı	20	400	380	20	95,00
Senaryo1	C	Manhattan Uzaklığı	20	400	390	10	97,50
Senaryo2	C	Manhattan Uzaklığı	20	400	360	40	90,00
Senaryo4	C	Manhattan Uzaklığı	20	400	372	28	93,00
Senaryo3	C	Manhattan Uzaklığı	20	400	380	20	95,00
Senaryo1	JavaScript	Kosinüs Benzerliği	20	400	390	10	97,50
Senaryo2	JavaScript	Kosinüs Benzerliği	20	400	382	18	95,50
Senaryo4	JavaScript	Kosinüs Benzerliği	20	400	370	30	92,50
Senaryo3	JavaScript	Kosinüs Benzerliği	5	25	25	0	100,00
Senaryo1	JavaScript	Öklid Uzaklığı	20	400	382	18	95,50
Senaryo2	JavaScript	Öklid Uzaklığı	20	400	382	18	95,50
Senaryo4	JavaScript	Öklid Uzaklığı	20	400	370	30	92,50
Senaryo3	JavaScript	Öklid Uzaklığı	5	25	24	1	96,00
Senaryo1	JavaScript	Manhattan Uzaklığı	20	400	390	10	97,50
Senaryo2	JavaScript	Manhattan Uzaklığı	20	400	380	20	95,00
Senaryo4	JavaScript	Manhattan Uzaklığı	20	400	370	30	92,50
Senaryo3	JavaScript	Manhattan Uzaklığı	5	25	24	1	96,00

Tüm diller için deneylerde birinci teknik olarak Kosinüs Benzerliği [41] kullanılmıştır. İkinci teknik olarak Öklid Uzaklığı [42] kullanılmıştır. Üçüncü teknik olarak Manhattan Uzaklığı [43] kullanılmıştır. Programlama dillerinin kendi içerisindeki çözümleri ile Kosinüs Benzerliği, Öklid Uzaklığı ve Manhattan Uzaklığı ile benzerliklerinin bulunması noktasında tek tek etiketleme yapılarak başarı oranları hesaplanmıştır.

5.1. C# Programlama Dilinde Kod Klon Tespiti

4 farklı senaryo için üretilmiş çözümlerden rastgele 226 farklı veri seçilmiştir. Senaryo1 için 58 dosya seçilmiştir. Senaryo2 için 54 dosya seçilmiştir. Senaryo3 için 54 dosya seçilmiştir. Senaryo4 için 60 dosya seçilmiştir. Benzerliklerin tespiti için ön işleme sonucu 34 metrik veri seti ortaya çıkartılmıştır. Üç farklı teknik ile çözümler analiz edilerek kod klon tespiti için deneyler yapılmıştır. Her bir dosya için birer birer diğer dosyalar ile metrikler üzerinden benzerlik analizi yapılmıştır. Bu analiz sonuçları karşılaştırmalı olarak Tablo 5.1'de verilmiştir. Bu deneylerden sonra yeni dosyalar eklenerek bir deney daha yapılmıştır. Tüm senaryolar için toplanan toplamda 226 dosyaya 89 dosya daha eklenerek toplamda tüm senaryolar için 315 dosya ile yapılan analiz sonuçları da Tablo 5.2'de verilmiştir.

Tablo 5.2 C# İçin Kosinüs benzerliği, Öklid uzaklığı ve Manhattan uzaklığı tekniklerinin kod klon tespit başarı oranları

	Başarı Oranları
Kosinüs Benzerliği Başarı Oranı (Eşik Değeri > 70)	%71,99
Öklid Uzaklığı Başarı Oranı (Eşik Değeri < 500)	%81,93
Manhattan Uzaklığı Başarı Oranı (Eşik Değeri < 1500)	%91,17

5.2. Java Programlama Dilinde Kod Klon Tespiti

Dört farklı senaryo için üretilmiş çözümlerden rastgele 160 farklı veri seçilmiştir. Senaryo1 için 40 dosya seçilmiştir. Senaryo2 için 40 dosya seçilmiştir. Senaryo3 için 40 dosya seçilmiştir. Senaryo4 için 40 dosya seçilmiştir. Benzerliklerin tespiti için ön işleme sonucu 34 metrik veri seti ortaya çıkartılmıştır. Üç farklı teknik ile çözümler analiz edilerek kod klon tespiti için deneyler yapılmıştır. Her bir dosya için birer birer diğer dosyalar ile metrikler üzerinden benzerlik analizi yapılmıştır. Bu analiz sonuçları

karşılaştırmalı olarak Tablo 5.1'de verilmiştir. Tüm senaryolar için toplanan dosyalar birleştirilerek toplamda 160 dosya ile yapılan deney sonuçları tablo 5.3'te verilmiştir.

Tablo 5.3 Java için Kosinüs benzerliği, Öklid uzaklığı ve Manhattan uzaklığı tekniklerinin kod klon tespit başarı oranları

	Başarı Oranları
Kosinüs Benzerliği Başarı Oranı (Eşik Değeri > 70)	%84,97
Öklid Uzaklığı Başarı Oranı (Eşik Değeri < 500)	%95,34
Manhattan Uzaklığı Başarı Oranı (Eşik Değeri < 1500)	%92,15

5.3. C Programlama Dilinde Kod Klon Tespiti

Dört farklı senaryo için üretilmiş çözümlerden rastgele 80 farklı veri seçilmiştir. Senaryo1 için 20 dosya seçilmiştir. Senaryo2 için 20 dosya seçilmiştir. Senaryo3 için 20 dosya seçilmiştir. Senaryo4 için 20 dosya seçilmiştir. Benzerliklerin tespiti için ön işleme sonucu 34 metrik veri seti ortaya çıkartılmıştır. Üç farklı teknik ile çözümler analiz edilerek kod klon tespiti için deneyler yapılmıştır. Her bir dosya için birer birer diğer dosyalar ile metrikler üzerinden benzerlik analizi yapılmıştır. Bu analiz sonuçları karşılaştırmalı olarak Tablo 5.1'de verilmiştir. Tüm senaryolar için toplanan dosyalar birleştirilerek toplamda 80 dosya ile yapılan deney sonuçları tablo 5.4'te verilmiştir.

Tablo 5.4 C için Kosinüs benzerliği, Öklid uzaklığı ve Manhattan uzaklığı tekniklerinin kod klon tespit başarı oranları

	Başarı Oranları
Kosinüs Benzerliği Başarı Oranı (Eşik Değeri > 70)	%94,7
Öklid Uzaklığı Başarı Oranı (Eşik Değeri < 500)	%98,4
Manhattan Uzaklığı Başarı Oranı (Eşik Değeri < 1500)	%98,4

5.4.JavaScript Programlama Dilinde Kod Klon Tespiti

Dört farklı senaryo için üretilmiş çözümlerden rastgele 65 farklı veri seçilmiştir. Senaryo1 için 20 dosya seçilmiştir. Senaryo2 için 20 dosya seçilmiştir. Senaryo3 için 5 dosya seçilmiştir. Senaryo4 için 20 dosya seçilmiştir. Benzerliklerin tespiti için ön işleme sonucu 34 metrik veri seti ortaya çıkartılmıştır. Üç farklı teknik ile çözümler analiz edilerek kod klon tespiti için deneyler yapılmıştır. Her bir dosya için birer birer diğer dosyalar ile metrikler üzerinden benzerlik analizi yapılmıştır. Bu analiz sonuçları

karşılaştırmalı olarak Tablo 5.1'de verilmiştir. Tüm senaryolar için toplanan dosyalar birleştirilerek toplamda 65 dosya ile yapılan deney sonuçları tablo 5.5'te verilmiştir.

Tablo 5.5 JavaScript için Kosinüs benzerliği, Öklid uzaklığı ve Manhattan uzaklığı tekniklerinin kod klon tespit başarı oranları

	Başarı Oranları
Kosinüs Benzerliği Başarı Oranı (Eşik Değeri > 70)	%94,37
Öklid Uzaklığı Başarı Oranı (Eşik Değeri < 500)	%92,88
Manhattan Uzaklığı Başarı Oranı (Eşik Değeri < 1500)	%93,25

5.5. Çapraz Programlama Dilleri ile Kod Klon Tespiti

Çapraz testler için öncelikle ikili diller arasında deneyler yapılmıştır. Yapılan deney sonuçları Tablo 5.6'da verilmiştir.

Tablo 5.6 Java, JavaScript, C ve C# Programlama dillerinin, Kosinüs benzerliği, Öklid uzaklığı ve Manhattan uzaklığı teknikleri ile klon tespit başarı oranları

Programlama Dilleri İkilileri	**Kosinüs Benzerliği**	**Öklid Uzaklığı**	**Manhattan Uzaklığı**
C# - Java	%79,08	%85,47	%84,51
C# - C	%61,77	%82,23	%85,99
C# - JavaScript	%64,27	%92,38	%92,51
Java - C	%81,06	%97,06	%97,01
Java - JavaScript	%78,50	%98,01	%98,01
C - JavaScript	%72,27	%91,36	%91,53

Son olarak da tüm dillerdeki veriler birleştirilerek deneyler yapılmıştır. Java, C#, C, JavaScript programlama dilleri üzerinde çapraz olarak kod klon tespiti üzerine deneyler yapılmıştır. Kosinüs benzerliği, Öklid uzaklığı ve Manhattan uzaklığı teknikleri ile benzerlikler oranları hesaplanmıştır. 620 dosya üç farklı teknik ile benzerlik oranları belirlenmiş Tablo 5.7'de verilmiştir. Kosinüs benzerliği tüm platformlarda yakın doğruluk oranıyla sonuçlanırken C programlama dilinde değerler birbirine paralel çıkmıştır. Java platformunda Öklid uzaklığı ön plana çıkarken JavaScript platformunda değerler yakın gözlemlenmiştir.

Tablo 5.7 Dört senaryoda programlama dilleri üzerinden Kosinüs benzerliği, Öklid uzaklığı ve Manhattan uzaklığının Ortalama Doğruluk Oranları

Teknik	C#	C	Java	JavaScript
Kosinüs Benzerliği (Eşik Değer > 70)	**%94,42**	**%93,88**	%95,08	**%95,54**
Öklid Uzaklığı (Eşik Değer < 500)	%83,03	**%93,88**	**%95,57**	%94,62
Manhattan Uzaklığı (Eşik Değer < 1500)	%81,43	**%93,88**	%94,22	%95,08

Bu çalışma ile Kosinüs benzerliği, Öklid uzaklığı ve Manhattan uzaklığı tekniklerinin C, C#, Java ve JavaScript programlama dilleri üzerinde kod benzerlik oranlarını tespit etmeye yönelik yaptığımız çalışmanın her teknik için ayrı ayrı kıyaslamaları oluşturulmuştur.

6. TARTIŞMA

Bu bölümde deneysel çalışmalarımızdan çıkarılan ana sonuçların bir özetini sunuyor, sonuçlarımızın etkilendiği koşulları detaylandırıyor ve gelecekteki araştırmalar için öneriler sunuyoruz.

6.1. Genel Bulgular

Deneysel bulgularımız, kullandığımız Kosinüs benzerliği, Öklid uzaklığı ve Manhattan uzaklığı yöntemlerinin özelliklerinden etkilendi. Karmaşıklık, boyut, sürdürülebilirlik ve kalite, ölçümler kullanılarak ölçülebilen kaynak kodun tüm yönleridir. Bu ölçümleri, her bir boyutun bir ölçüm yerine geçtiği yüksek boyutlu bir ortamda vektörler olarak görselleştirmek yaygın bir uygulamadır. Kosinüs benzerliği, söz konusu vektörler arasındaki açıyı hesaba katan vektörden bağımsız bir benzerlik ölçüsüdür. Kosinüs benzerliği, kod metriklerinde Öklid uzaklığı ve Manhattan uzaklığından daha iyi performans gösterir çünkü vektörlerin boyutuna duyarlı değildir ki bu önceki ikisi için bir sorun olabilir. Vektörlerin büyüklüğü, düşük boyutlu verilerde önemli bir rol oynar ve görüntü işleme ve bilgisayarla görme görevleri için Öklid uzaklığı ve Manhattan uzaklığı yöntemlerini tercih edilir hale getirir. Örneğin, görüntü işlemede, piksel değerleri tipik olarak düşük boyutlu bir vektör olarak temsil edilir ve Öklid uzaklığı ve Manhattan uzaklığı gibi mesafe ölçüleri, iki görüntünün karşılaştırılabilirlik derecesini değerlendirmek için kullanılabilir [44]. Çalıştığımız veri kümesinin nispeten mütevazı boyutuna rağmen, kodlar arasındaki benzerliği analiz ederken Kosinüs benzerliğinin diğer kod ölçümlerinden daha iyi performans gösterdiğini gördük çünkü yüksek boyutlu verileri işlemek için daha uygun. Öte yandan, düşük boyutlu veriler Öklid uzaklığı veya Manhattan uzaklığı kullanılarak analiz edilebilir.

6.2. Geçerliliğe Yönelik Tehditler

Bu çalışmanın bulguları, herhangi bir araştırma çalışmasında olduğu gibi, geçerliliğe yönelik bir takım potansiyel tehditler nedeniyle dikkatle yorumlanmalıdır. Bu tehditler iki farklı kategoriye ayrılabilir: iç ve dış.

6.3. İçsel Geçerliliğe Yönelik Tehditler

Benzerlik ölçümlerinin sonuçları, kodu temizlemek için kullanılan ön işleme yöntemlerinden etkilenebilir. Örneğin, belirli yorumlar veya biçimlendirme koddan

silinirse benzerlik ölçüleri her iki yönde de çarpık olabilir. Mümkün olduğu kadar doğru olması için, deneylerde kullanılan koddan yorumları kaldırmadık; ancak sonuçlar, geliştiricilerin değişen oranlarda yorum ve açıklama yazma eğiliminde olmasından etkilenebilir.

Benzerlik ölçümleri, kodu açıklamak için seçilen özelliklerden etkilenebilir. Bazı programlama dilleri veya kod yapıları için daha önemli olan özelliklerin atlanması, yanlış benzerlik sonuçlarına neden olabilir. Bu olasılığı azaltmak için deneylerimizi sadece yaygın olarak kullanılan yazılım metrikleri ile yaptık. Çeşitli programlama dillerine özgü metrikler olduğu için bu yola gittik.

Bu araştırmada kullanılan üç benzerlik ölçüsü olan Kosinüs benzerliği, Öklid uzaklığı ve Manhattan uzaklığının her koşulda en iyi seçenek olmaması mümkündür. Çeşitli programlama dilleri ve kod yapıları, diğer benzerlik ölçümlerinden daha fazla yararlanabilir. Araştırmamız sayesinde, Kosinüs Benzerliği ölçeğinde 70, Öklid Uzaklığı ölçeğinde 500 ve Manhattan Uzaklığı ölçeğinde 1500 eşik değerlerinin optimal olduğunu bulduk. Ancak, bu eşik değerleri daha iyi optimize edilerek daha yüksek kod klon saptama doğruluğu elde edilebilir.

Bu analizde alınan kod ön işleme, özellik seçimi ve benzerlik ölçüm kararlarının yanı sıra kaynak kodun kendisinin belgelenmesi, çalışmanın tekrarlanabilirliği için çok önemlidir.

6.4. Dışsal Geçerliliğe Yönelik Tehditler

Bu çalışmanın örneklem büyüklüğü, bu dört dili kullanan tüm programcıların göstergesi olmayabilir. Örneklerin daha deneyimli programcılara veya belirli bir programlama dilini veya kütüphane setini tercih edenlere yönelik olması mümkündür. Koleksiyonda kullanılacak kodlar, dikkatli bir değerlendirmeden sonra seçildi. Değerlendirmeler, hangi seçeneklerin en çok tavsiye edildiğine dair öngörü sağladı. Bu yollarla, her bir programlama dili için mevcut olan en iyi kaynaklardan örnekler toplamaya çalıştık.

Bu araştırmanın sonuçlarının diğer programlama dillerine veya başka amaçlarla veya farklı ortamlarda yazılmış programlara uyarlanamaması mümkündür. Sonuç olarak, bulgular diğer bağlamlara genellenemez ve bulgular yalnızca bu belirli alana uygulanabilir.

6.5. Gelecekteki Yönlendirmeler

Daha geniş ve daha çeşitli bir programcı örneği, çoklu ön işleme yöntemleri ve özellik kümeleri ve farklı benzerlik ölçümlerinin bulgularının karşılaştırılması, geçerliliğe yönelik bu tehditleri hafifletmeye yardımcı olmak için gelecekteki çalışmalarda kullanılabilir. Tekrar üretilebilirlik, açık kaynak kodunun kullanılması ve tüm prosedürlerin ve bulguların kapsamlı bir şekilde kaydedilmesi yoluyla da geliştirilebilir.

Understand uygulaması çok çeşitli bilgisayar dillerini desteklese de onu veri toplamamız gereken tüm diller için kullanamadık. Örneğin, Python desteğinin olmaması, onu deneylerimizde kullanamayacağımız anlamına geliyordu. Öte yandan, yazılım ölçümleri manuel çıkarım yoluyla toplanırsa, bu kısıtlama atlanabilir ve çalışmanın genel kapsamı genişletilebilir.

Önerdiğimiz yaklaşımların yanı sıra Jeccard ve Ngram benzerlik analizlerini birleştirmek mümkündür. Seyrek verileri işleme konusundaki potansiyel zayıflıkları ve koddaki küçük değişikliklere duyarlılıkları nedeniyle, deneylerimizden Jaccard ve Ngram benzerlik yaklaşımlarını çıkardık.

7. SONUÇLAR

Modern uygulamaların sürekli artan karmaşıklığı ile yazılım tasarımı her zaman geliştirilmektedir. Bir mikroservis mimarisini benimsemek yönetimi basitleştirebilse de özellikle çok dilli mikroservislerde kod tekrarını tespit etmeyi zorlaştırabilir. Yazılım sistemlerinde, kod klonları, bakım için gereken zaman ve kaynak miktarını artırabilen özdeş kod bölümleridir. Bu nedenle, kodda herhangi bir değişiklik yapmadan veya kopyaları kaldırmadan önce kod klonlarını tespit etmek çok önemlidir.

Araştırmalarımız ile yazılım dünyasındaki zaman kaybının düşürülmesine, klon kodlardan kaynaklı bakım maliyetlerinin azaltılmasına ve iş gücü kaybının azaltılmasına, hatalı kodların kopyalanarak çoğaltılmasında kod klonların etkisinin azaltılmasına, illegal kod klonların tespit edilmesine, üçüncü parti sistemlerden ve açık kaynak kodlu projelerden kopyalanarak oluşturulan ve güvenlik zafiyeti içeren klon kodlardan kaynaklı riskli durumların oluşmasının engellenmesine katkı sağlanması hedeflenmiştir.

Bu araştırma JavaScript, C, C# ve Java ile geliştirilen yazılım bileşenlerinin Kosinüs Benzerliği, Öklid Uzaklığı ve Manhattan Uzaklığı ölçümlerini kullanarak birbirlerinin kod klonları olma derecesini analiz etmek için bir metodoloji sunmuştur. Benzer kodların tespit edilmesinde üç farklı teknik ile deneyler gerçekleştirilmiştir. 4 farklı senaryo üzerindeki çözümlere odaklanılmıştır. Toplamda 3000'den fazla kaynak kodu içeren dosya incelenmiş ve deneylere konu olacak şekilde 620 dosya seçilmiştir. Seçilen tüm dosyalar tek tek incelenerek 34 metrik verisi ortaya çıkartılmıştır. Her dosya yazıldığı programlama dili içerisinde kümelenmiş ve kümelemelerden kartezyen çarpımlar ile karşılaştırılmak üzere ikililer oluşturulmuştur. Veriler seçilen dört farklı senaryo için farklı yazılım geliştiriciler tarafından oluşturulmuş ve cevapların doğrulukları bir otorite tarafından onaylanmıştır. C#, C, JavaScript ve Java programlama dilleri ile oluşturulmuş çözümler derlenerek veri kümesi oluşturulmuştur. Dört farklı programlama için ve dört farklı senaryo kendi içlerinde olacak şekilde deneyler yapılmıştır. Bu senaryolar ve açıklamaları şöyledir; Senaryo1 (Children and Candies): Çocuklar ve şekerler üzerine oluşturulmuş bir senaryodur. Bir anaokulunda dağıtılan şeker sayılarını bulmaya yönelik bir matematiksel problemin yazılımsal olarak çözümlenmesi istenmiştir. Oluşturulan kaynak kodlar toplanarak

birinci senaryoya veri oluşturulmuştur. Senaryo ile ilgili detaylı açıklamalar ve kısıtlar EK A.1'de detaylı olarak verilmiştir. Senaryo2 (Iroha Loves Strings): Metinsel bir dizi içeriğinin belirlenen kısıtlar içerisinde bir sıra ile eklenerek oluşturduğu küme içerisinden bir dizi bulunmasına yönelik bir matematiksel problemin yazılımsal olarak çözümlenmesi istenmiştir. Çözümler için farklı yazılım geliştiriciler tarafından oluşturulan kaynak kodlar toplanarak ikinci senaryoya veri oluşturulmuştur. Senaryo ile ilgili detaylı açıklamalar ve kısıtlar EK A.2'de detaylı olarak verilmiştir. Senaryo3 (Tak and Cards) : kartlar ve bu kartların rastgele seçilmesi ile oluşan ortalama sayılar üzerinden kaç farklı seçimin olacağını bulmaya yönelik bir matematiksel problemin yazılımsal olarak çözümlenmesi istenmiştir. Çözümler için farklı yazılım geliştiriciler tarafından oluşturulan kaynak kodlar toplanarak birinci senaryoya veri oluşturulmuştur. Senaryo ile ilgili detaylı açıklamalar ve kısıtlar EK A.3'te detaylı olarak verilmiştir. Senaryo4 (Many Formulas):1 ve 9 rakamları arasından verilen bir sayı dizi ile toplama(+) işaretinin birleştirilmesi ve senaryo detayında verilen kısıtlara uygun bir şekilde olası eklenebilecek tüm eklemelerin yapılması sonucunda oluşabilecek toplamı bulmaya yönelik bir matematiksel problemin yazılımsal olarak çözümlenmesi istenmiştir. Çözümler için farklı yazılım geliştiriciler tarafından oluşturulan kaynak kodlar toplanarak dördüncü senaryoya veri oluşturulmuştur. Senaryo ile ilgili detaylı açıklamalar ve kısıtlar EK A.4'te detaylı olarak verilmiştir.

Deney yoluyla, çeşitli yaklaşımlar için deneysel eşikler belirledik ve %91,59 doğruluk oranıyla Manhattan Uzaklığı yaklaşımının platformlar arası kod klonlarını tespit etmede en etkili yaklaşım olduğunu belirledik. Burada açıklanan araştırma metodolojisi daha geniş bir uygulanabilirliğe sahiptir. Örneğin, bu yöntem, yazılım geliştirme ekipleri tarafından kod tabanının modüler hale getirilme, kapsülleme ve yazılım mühendisliği en iyi uygulamalarında kodun yeniden kullanımını değerlendirmek için kullanılabilir. Ekip, sistemlerinin çeşitli bileşenleri arasındaki benzerlik derecesini analiz ederek gelecekteki geliştirmeler için kaynakları daha iyi tahsis edebilir ve büyüme alanlarını belirleyebilir. Ayrıca, burada sağlanan metodolojinin bu analizde kullanılan programlama dillerine özgü olmadığını vurgulamak önemlidir. Geliştiriciler bu yaklaşımı kullanarak, sistemlerinin kod klonlarından arınmış olmasını sağlayabilir, böylece hata riskini azaltabilir ve sistemlerini daha güvenilir ve verimli hale getirebilir.

KAYNAKÇA

[1] "October Github," [Online]. Available: https://octoverse.github.com/.

[2] C. K. R. &. J. R. Cordy, "https://www.researchgate.net/publication/277287260_A_Survey_on_Software_Clone_Detection_Research," [Online]. Available: https://www.researchgate.net/publication/277287260_A_Survey_on_Software_Clone_Detection_Research.

[3] S. Hassan and R. Bahsoon,, "Microservices and their design trade-offs: A self-adaptive roadmap," in *2016 IEEE International Conference on Services Computing (SCC)*, San Francisco, CA, USA, 2016.

[4] J. S. a. C. Roy., "Evaluating clone detection tools with bigclonebench. ICSME'15," 2015.

[5] Yueming Wu, Deqing Zou, Shihan Dou, Siru Yang, Wei Yang, Feng Cheng, Hong Liang, Hai Jin , "ASE '20: Proceedings of the 35th IEEE/ACM International Conference on Automated Software EngineeringDecember 2020 Pages".

[6] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue., "CCFinder: a multilinguistic token-based code clone detection system for large scale source code.," 2002.

[7] Hitesh Sajnani, Vaibhav Saini, Jeffrey Svajlenko, Chanchal K Roy, and Cristina V, "SourcererCC: Scaling code clone detection to big-code. In Proceedings of the 38th International Conference on Software Engineering. IEEE, 1157-1168.," 2016.

[8] Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stephane Glondu., "Deckard: Scalable and accurate tree-based detection of code clones.," 2007.

[9] Huihui Wei and Ming Li, "Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code. In Proceedings of the 26th International Joint Conference on Artificial Intelligence. 3034-3040," 2017.

[10] Gang Zhao and Jeff Huang, "Deepsim: deep learning code functional similarity. In Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, 141-151.," 2018.

[11] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Kaixuan Wang, and Xudong Liu, "A novel neural source code representation based on abstract

syntax tree. In Proceedings of the 41st International Conference on Software Engineering IEEE, 783-794," 2019.

[12] N. Davey, P. Barson, S. Field, R. Frank, and D. Tansley, "The development of a software clone detector. IJAST, 1(3/4),," 1995.

[13] J. Bailey ve E. Burd, "Evaluating clone detection tools for use during preventative maintenance. SCAM'02.".

[14] M. Rieger, " Effective clone Detection Without LanguageBarriers. PhD thesis,," 2005.

[15] A. Walenstein and A. Lakhotia, "The software similarity problem in malware analysis. Dagstuhl Seminar Proceedings," 2007.

[16] L. Jiang, Z. Su, and E. Chiu, "Context-based detection of clone-related bugs. ESEC/FSE'07.".

[17] Lucia, D. Lo, L. Jiang, and A. Budi, "Active refinement of clone anomaly reports. ICSE'12.".

[18] X. Wang, Y. Dang, L. Zhang, D. Zhang, E. Lan, and H. Mei., "Can i clone this piece of code here? ASE'12.".

[19] Y. Dang, D. Zhang, S. Ge, C. Chu, Y. Qiu, and T. Xie. Xiao, "Tuning code clones at hands of engineers in practice. ACSAC'12.".

[20] N. Milea, L. Jiang, and S. Khoo, "Scalable detection of missed cross-function refactorings. ISSTA'14.".

[21] N. Milea, L. Jiang, and S. Khoo, "Vector abstraction and concretization for scalable detection of refactorings. FSE'14.".

[22] Martin White, Michele Tufano, Christopher Vendome, and Denys Poshyvanyk., "Deep learning code fragments for code clone detection.," 2016.

[23] Jiachen Yang, Keisuke Hotta, Yoshiki Higo, Hiroshi Igaki, and Shinji Kusumoto., "Classification model for code clones based on machine learning. Empirical Software Engineering 20, 4 (2015), 1095–1125.," 2015.

[24] Li ping Zhang; Dong sheng Liu, "AST-based multi-language plagiarism detection method," in *2013 IEEE 4th International Conference on Software Engineering and Service Science*, Beijing, China, 2013.

[25] Yue Zou; Bihuan Ban; Yinxing Xue; Yun Xu, "CCGraph: a PDG-based code clone detector with approximate graph matching," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Melbourne, VIC, Australia, 2020.

[26] Astrit Desku; Bujar Raufi; Artan Luma; Besnik Selimi, "Cosine Similarity through Control Flow Graphs For Secure Software Engineering," in *2021 International Conference on Engineering and Emerging Technologies (ICEET)*, Istanbul, Turkey, 2021.

[27] David, Y., & Yahav, E., "Tracelet-based code search in executables, in.," 2014.

[28] Daniel Perez, Shigeru Chiba, "Cross-language clone detection by learning over abstract syntax trees," in *MSR '19: Proceedings of the 16th International Conference on Mining Software Repositories*, 2019.

[29] Eschweiler, S., Yakdan, K., & Gerhards-Padilla E., "discovre: Efficient crossarchitecture identification of bugs in binary code, in.," 2016.

[30] Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stephane Glondu, "Deckard: Scalable and accurate tree-based detection of code clones. In Proceedings of the 29th International Conference on Software Engineering. IEEE, 96-105.," 2007.

[31] [Online]. Available: https://msdn.microsoft.com/library/system.codedom.aspx.

[32] Shalev, N., & Partush, N., "Binary similarity detection using machine learning," 2018.

[33] Feng, Q., Zhou, R., Xu, C., Cheng, Y., Testa, B., & Yin, H. , "Scalable graph-based bug search for firmware images.," 2016.

[34] Raghavan Komondoor and Susan Horwitz. , "Using slicing to identify duplication in source code.," 2001.

[35] "Understand," [Online]. Available: https://www.scitools.com/.

[36] "https://atcoder.jp/," [Online]. Available: https://atcoder.jp/.

[37] "https://atcoder.jp/contests/abc043/tasks/abc043_a," [Online]. Available: https://atcoder.jp/contests/abc043/tasks/abc043_a.

[38] "https://atcoder.jp/contests/abc042/tasks/abc042_b," [Online]. Available: https://atcoder.jp/contests/abc042/tasks/abc042_b.

[39] "https://atcoder.jp/contests/abc044/tasks/arc060_a," [Online]. Available: https://atcoder.jp/contests/abc044/tasks/arc060_a.

[40] "https://atcoder.jp/contests/abc045/tasks/arc061_a," [Online]. Available: https://atcoder.jp/contests/abc045/tasks/arc061_a.

[41] Dawei Xue; Yong Wang, "Applying Cosine Similarity to Discount Evidence," in *2017 10th International Symposium on Computational Intelligence and Design (ISCID)*, Hangzhou, China, 2017.

[42] M. D. Malkauthekar, "Analysis of euclidean distance and Manhattan Distance measure in face recognition," in *Third International Conference on Computational Intelligence and Information Technology (CIIT 2013)*, Mumbai, 2013.

[43] "https://www.sciencedirect.com/topics/mathematics/manhattan-distance," [Online]. Available: https://www.sciencedirect.com/topics/mathematics/manhattan-distance.

[44] Ali Seyed Shirkhorshidi, Saeed Aghabozorgi, Teh Ying Wah, "A Comparison Study on Similarity and Dissimilarity Measures in Clustering Continuous Data," 2015.

EKLER

EK A.1 Senaryo1 (Children and Candies ABC Edit) Detayı

Açıklama

AtCoder Anaokulunda N çocuk var. Evi Bey çocukları sıraya dizer, ardından sıradaki ilk çocuğa 1 şeker, ikinci çocuğa 2 şeker, ..., N. çocuğa N şeker verir. Toplamda kaç şeker gerekli olacak?

Kısıtlar

$1 \leqq N \leqq 100$

EK A.2 Senaryo2 (Iroha Loves Strings ABC Edition) Detayı

Açıklama

Iroha'nın $S_1,S_2,...,S_N$. Elemanları olan N dizisi vardır. Her bir dizinin uzunluğu L'dir.

Uzun bir dizi oluşturmak için tüm dizileri bir sırayla birleştirecektir.

Bu şekilde üretebileceği tüm dizeler arasında sözlüksel olarak en küçük olanı bulun.

$s=s_1s_2s_3...s_n$ dizisi sözlüksel olarak diğer bir dizi olan $t=t_1t_2t_3...t\_m$ dizisinden daha küçüktür. Ancak ve ancak aşağıdakilerden biri geçerliyse:

$s_j=t_j$ öyle ki $j(1\leqq j<i)$ tüm indeksler için ve $s_i<t_i$. İçin $i(1\leqq i\leqq min(n,m))$ böyle bir dizin mevcuttur.
$s_i=t_i$ tüm sayılar için $i(1\leqq i\leqq min(n,m))i(1\leqq i\leqq min(n,m))$, ve $n<mn<m$.

Kısıtlar

$1\leqq N,L\leqq 100$

Her i için , S_i'nin uzunluğu = L

Her i için, S_i küçük harflerden oluşur.

EK A.3 Senaryo3 (Tak and Cards) Detayı

Açıklama

Tak'ın N kartı var. i'nci ($1 \leq i \leq N$) kart x_i bir tamsayıdır. Tak bu N karttan bir veya daha fazla kart seçiyor, böylece seçilen kartlara yazılan tam sayıların ortalaması tam olarak A oluyor. Seçimini kaç farklı şekilde yapabilir?

Kısıtlar

$1 \leq N \leq 50$

$1 \leq A \leq 50$

$1 \leq x_i \leq 50$

N, A, x_i birer tamsayıdır.

EK A.4 Senaryo4 (Many Formulas) Detayı

Açıklama

Size 1(dahil) ile 9(dahil) arasında rakamlardan oluşan bir S dizisi verilir. Bu dizide iki harf arasındaki bazı konumlara (muhtemelen hiçbiri) + işaretini ekleyebilirsiniz. Eklemelerden sonra art arda + oluşmamalıdır. Bu şekilde elde edilebilecek tüm diziler formül olarak değerlendirilebilir. Olası tüm formülleri değerlendirin ve sonuçların toplamını yazdırın.

Kısıtlar

$1 \leq |S| \leq 9$

S dizisindeki tüm değerler 1 ile 9 arasındaki rakamlardan oluşmaktadır.

EK B.1 Senaryo1 (Children and Candies ABC Edit) Kod Örnekleri

Örnek1:

```
using System;

namespace candy
{
    class Program
    {
        static void Main(string[] args)
        {
            int n;
            int s;
            int i;

            n = int.Parse(Console.ReadLine());

            s = 0;
            for (i = 1; i <= n; i++)
                s += i;

            Console.WriteLine(s);
        }
    }
}
```

Örnek2:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace AtCoder_A
{
    class Program
    {
        static void Main(string[] args)
        {
            int N = int.Parse(Console.ReadLine());
            int ans = 0;
            for (int i = 1; i <= N; i++)
            {
                ans += i;
            }
            Console.WriteLine(ans);
        }
    }
}
```

EK B.2 Senaryo2 (Iroha Loves Strings ABC Edition) Kod Örnekleri

Örnek1:

```
using System;
using System.Collections.Generic;
using System.Linq;

class Program
{
    static void Main()
    {
        var nl = Console.ReadLine().Split().Select(int.Parse).ToArray();
        var s = new List<string>();
        for (int i = 0; i < nl[0]; i++)
            s.Add(Console.ReadLine());

        s.Sort();
        foreach (var i in s)
            Console.Write(i);
        Console.WriteLine();
    }
}
```

Örnek2:

```
using System;
using System.Linq;
using System.Collections.Generic;
namespace test_1
{
    class Practice
    {

        static void Main(string[] args)
        {
            int[] n = Console.ReadLine().Split(" ").Select(a =>
int.Parse(a)).ToArray();
            List<string> s = new List<string>();
            for (int i = 0; i < n[0]; i++)
            {
                s.Add(Console.ReadLine());
            }
            s.Sort();
            foreach (string i in s)
            {
                Console.Write(i);
            }
            Console.WriteLine();
        }
    }
}
```

EK B.3 Senaryo3 (Tak and Cards) Kod Örnekleri

Örnek1:

```
using System;

public class Hello
{
    public static void Main()
    {
        string[] line = Console.ReadLine().Trim().Split(' ');
        var n = int.Parse(line[0]);
        var a = int.Parse(line[1]);
        line = Console.ReadLine().Trim().Split(' ');
        var x = Array.ConvertAll(line, int.Parse);
        var dp = new long[n + 1, n + 1, n * 50 + 1000];
        dp[0, 0, 0] = 1;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                for (int k = 0; k <= n * 50 + 1; k++)
                {
                    dp[i + 1, j, k] += dp[i, j, k];
                    dp[i + 1, j + 1, k + x[i]] += dp[i, j, k];
                }
        var rep = 0L;
        for (int i = 1; i < n + 1; i++)
            rep += dp[n, i, i * a];
        Console.WriteLine(rep);
    }
}
```

Örnek2:

```
using System;
using System.Collections.Generic;
using System.Linq;
using static System.Console;
using static System.Convert;
using static System.Math;

class Program
{
    static void Main(string[] args)
    {
        var na = Array.ConvertAll(ReadLine().Split(' '), int.Parse);
        var x = Array.ConvertAll(ReadLine().Split(' '), int.Parse);
        var nx = na[0] * Max(x.Max(), na[1]);
        var dp = new long[na[0] + 1, 2 * nx + 1];
        dp[0, nx] = 1;
        for (var i = 0; i < na[0]; i++)
            x[i] -= na[1];
        for (var i = 0; i <= na[0]; i++)
        {
            for (var j = 0; j <= 2 * nx; j++)
            {
                if (i == 0 && j != nx) { dp[i, j] = 0; continue; }
                else if (i == 0) continue;
                var dif = j - x[i - 1];
                if (dif < 0 || dif > 2 * nx) dp[i, j] = dp[i - 1, j];
                else dp[i, j] = dp[i - 1, j] + dp[i - 1, dif];
            }
        }
        WriteLine(dp[na[0], nx] - 1);
    }

    private const string ALFA = "abcdefghijklmnopqrstuvwxyz";
    private const int MOD = 1000000007;
}
```

EK B.4 Senaryo4 (Many Formulas) Kod Örnekleri

Örnek1:

```
using System;
using System.Linq;
using System.Collections.Generic;
using static System.Console;
using System.Text;
using System.IO;
using static System.Math;
namespace AtCoder
{
    class Program
    {
        static public long[] Sarray() { return
ReadLine().Split().Select(long.Parse).ToArray(); }
        static public List<long> Slist() { return
ReadLine().Split().Select(long.Parse).ToList(); }
        static long Mod = (long)1e9 + 7;
        static void Main(string[] args)
        {
            //SetOut(new StreamWriter(OpenStandardOutput()) { AutoFlush =
false });
            var S = ReadLine();
            var N = S.Length;
            var all = (1 << (N - 1));
            var ans = 0L;
            for (var i = 0; i < all; ++i)
            {
                var mask = 1L;
                var plusPos = new List<int>() { 0 };
                for (var j = 0; j < N - 1; ++j, mask <<= 1)
                {
                    if ((i & mask) != 0)
                        plusPos.Add(j + 1);
                }
                plusPos.Add(N);
                for (var j = 0; j < plusPos.Count - 1; ++j)
                {
                    var subS = S.Substring(plusPos[j], plusPos[j + 1] -
plusPos[j]);
                    ans += long.Parse(subS);
                }
            }
            WriteLine(ans);
            //Out.Flush();
        }
    }
}
```

Örnek2:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.ComponentModel.Design;
using System.IO.Compression;
using System.Linq;
using System.Numerics;
using System.Text;
using Console = System.Console;

namespace Atcoder
{
    class Program
    {
        private const long COMDIV = 1000000007;

        static void Main(string[] args)
        {
            var S = Console.ReadLine();
            // {0, 1, ..., n-1} の部分集合の全探索
            int n = S.Length - 1;
            long sum = 0;
            for (int bit = 0; bit < (1 << n); ++bit)
            {
                string part = S[0].ToString();
                for (int i = 0; i < n; ++i)
                {
                    if ((bit & (1 << i)) != 0)
                    {
                        sum += long.Parse(part);
                        part = S[i + 1].ToString();
                    }
                    else
                    {
                        part += S[i + 1];
                    }
                }

                sum += long.Parse(part);
            }

            Console.WriteLine(sum);
        }
    }
}
```