Android'de Çökme Tespitini İyileştirme Amaçlı Model-Tabanlı ve Rastgele Karma Yöntem

Yavuz Köroğlu[1], Mustafa Efendioğlu[1], ve Alper Şen[1]

Boğaziçi Üniversitesi, Bilgisayar Mühendisliği Bölümü
{yavuz.koroglu,mustafa.efendioglu,alper.sen}@boun.edu.tr

Özet. Android uygulamaları dünya çapında yaygın olarak kullanılmaktadır. Bu uygulamaların birçoğu potansiyel çökmeler (crash) içermektedir. Bu çökmeleri tespit etmek amaçlı olarak son yıllarda Android uygulamalarının kara-kutu (black-box) testini yapan birçok akademik çalışma gerçekleştirilmiştir. Test amaçlı rastgele eylemler seçen basit bir araç olan Monkey, en gelişkin test araçlarından daha fazla çökme tespit etmekte, fakat uygulamaların derinliklerindeki aktivitelere ulaşamamaktadır. Bu çalışmamızda model öğrenen AndroFrame aracımızı Monkey ile karma olarak çalıştırarak daha çok aktiviteyi kapsayıp çökme tespit performansını iyileştirmeyi amaçlamaktayız. Karma yöntemimiz 20 adet Android uygulaması içerisinde AndroFrame'e kıyasla %2 daha fazla aktivite kapsamasına ulaşıp fazladan 21 çökme tespit etmiştir. Monkey'e kıyasla ise %24 daha fazla aktivite kapsamasına ulaşıp fazladan 5 çökme tespit etmiştir.

Anahtar Kelimeler: Mobil Uygulama Testi, Grafiksel Kullanıcı Arayüzü Testi, Otomatik Test Yaratımı

Combining Model-Based and Random Approaches to Improve Crash Detection in Android

Yavuz Köroğlu[1], Mustafa Efendioğlu[1], and Alper Şen[1]

Bogazici University, Department of Computer Engineering
{yavuz.koroglu,mustafa.efendioglu,alper.sen}@boun.edu.tr

Abstract. Android applications are widely used around the world. Most of these applications contain potential crashes. Many recent academic studies focus on black-box testing of Android applications to detect these crashes. A simple random testing tool, Monkey, detects more crashes than the state-of-the-art black-box testing tools, but can not reach some

activities that are located deep within the application. We propose an hybrid approach that combines our model-learning tool, AndroFrame, and Monkey. With this hybrid approach, we aim to increase activity coverage and improve crash detection. We conduct experiment on 20 Android applications. As a result, our hybrid approach achieves 2% more activity coverage and detects 21 more crashes compared to AndroFrame. Compared to Monkey, our hybrid approach achieves 24% more coverage and detects 5 more crashes.

Keywords: Mobile Application Testing, GUI Testing, Automated Test Generation

1 Giriş

Mobil uygulamalar günlük hayatımızın vazgeçilmez parçalarından biri olmuşlardır. İstatistikler bir kişinin günde ortalama 3 saat mobil telefon kullandığını ve bu sürenin de %90'ını mobil uygulamalara harcadığını gösteriyor [3]. Büyüyen mobil uygulama marketi trendini takiben, üst düzey test konferans ve yayınlarında mobil uygulama testi yayınları da artmaktadır [12]. Mobil uygulama pazarında Android uygulamaları en büyük paya sahiptir. Bu çalışmada Android Grafiksel Kullanıcı Arayüzü (GKA) testine odaklanmaktayız.

Son on yılda A^3E [1], Dynodroid [9], PUMA [7], ve SwiftHand [4] gibi birçok akıllı otomatik test yaratım aracı geliştirilmiştir. Fakat, basit bir rastgele test yaratım aracı olan Monkey'in, bunların hepsinden daha çok çökme tespiti yaptığı gözlemlenmiştir [11]. Bunun nedeni, Monkey'in diğer araçlara kıyasla çok daha fazla çeşit olay test edebiliyor olmasıdır [11]. Monkey'in kötü olduğu taraf ise Test Altındaki Uygulamanın (TAU) derinliklerindeki ekranlara ulaşamamasıdır. Bu yüzden Dynodroid gibi araçlar daha az çökme tespit etmesine [11] rağmen Monkey'e kıyasla daha büyük aktivite kapsamasına ulaşabilmektedir [9].

Bu çalışmamızda hem aktivite kapsaması, hem de çökme tespiti bakımından iyi bir performans elde etmek amacıyla AndroFrame-Monkey (AFM) adını verdiğimiz karma bir otomatik test aracı önermekteyiz. AndroFrame [8], Android platformu için geliştridiğimiz tam otomatik bir kara-kutu test yaratım aracıdır. AndroFrame daha önce dizayn edilmiş Android Grafiksel Kullanıcı Arayüzü (GKA) test araçlarının özelliklerini birleştirmektedir; A^3E'de [1] olduğu gibi bütün giriş noktalarını (exportable activity) kapsamakta, SwiftHand'de [4] olduğu gibi TAU'nun sonlu-durum modelini çıkarmakta ve PUMA'da [7] önerildiği gibi, farklı durumları ayırt edebilmek için kosinüs benzerliğini kullanmaktadır.

AndroFrame aktivite kapsaması bakımından diğer araçlar arasında en iyi, çökme tespiti bakımından ise Monkey ile aşağı yukarı eşittir. AFM aracı, AndroFrame ile çıkarılan TAU yaklaşık bir modeli üzerinden tüm aktiviteleri ziyaret edecek en kısa yolları bulup, bu yollarla ziyaret ettiği aktivitelerin her birinde Monkey aracını ayrı ayrı çalıştırarak çökme tespiti yapmaktadır. Bildiğimiz kadarıyla, AFM, Monkey ile bir başka otomatik test yaratım aracını bir arada kullanan ilk çalışmadır.

Makalenin geri kalanı şu şekilde düzenlenmiştir. Bölüm 2, Android sistem altyapısı hakkında gerekli bilgileri vermektedir. Bölüm 3, detaylı bir akış çizelgesi ve basit bir örnek üzerinden AFM aracının çalışma prensiplerini anlatmaktadır. Bölüm 4, AFM aracının Monkey ve AndroFrame araçları ile kıyaslanması için oluşturduğumuz altyapıyı ve deney sonuçlarını açıklamaktadır. Bölüm 5, çalışmamızın geçerliliği ile ilgili konuları ve dizayn seçimlerimiz hakkında açıklamalar içermektedir. Bölüm 6, ilgili çalışmaları açıklamaktadır. Bölüm 7, çalışmamızı özetleyip gelecek çalışmalar için olası yolları anlatmaktadır.

2 Android Sistem Altyapısı

Bu bölümde, kullandığımız yöntemlerin kolay anlaşılabilmesi için Android Grafiksel Kullanıcı Arayüzü (GKA) ve Android uygulamaları hakkında temel bilgiler vermekteyiz.

Android GKA, aktivite (activity) ve olay (event) tabanlıdır. Aktiviteler bir ekrandaki kullanıcı arayüzünü temsil eder ve GKA bileşenlerinden (widget) oluşur. Her bir GKA bileşeni (örn. düğme veya metin girdisi), piksel cinsinden bileşenin sınır koordinatlarını (x_1, y_1, x_2, y_2) tanımlayan ve kullanıcının bileşenle hangi eylemler (action) aracılığıyla etkileşime girebileceğini belirten bir takım özelliklere sahiptir. Bu özelliklere, *tür*, *etkin* (enabled), *tıklanabilir* (clickable), *uzun tıklanabilir* (longclickable), *kaydırılabilir* (scrollable), ve *şifre* (password) örnek olarak verilebilir.

Bir kullanıcı, Android sistemi ile GKA bileşenleri üzerinden *olaylar* aracılığı ile etkileşime girebilir. Olayları temel olarak iki kategoriye ayırabiliriz, *sistem olayları* ve *GKA olayları* (GKA eylemleri). Tipik olarak literatürde kullanılanlardan daha kapsamlı bir GKA eylemleri listesini Tablo 1'de göstermekteyiz. Eylemler üç kategoriden oluşmaktadır; bağlamsal olmayan (non-contextual), bağlamsal (contextual) ve özel (special). Bağlamsal olmayan eylemler kullanıcı hareketleriyle tetiklenen eylemlerdir. *Tıklama* ve *uzun tıklama* eylemleri, tıklanılacak x ve y koordinatları olmak üzere iki parametre alırlar. *Metin girdisi* eylemi x, y koordinatları ve girilecek metni belirten üç parametre alır. *Kaydırma* eylemi beş parametre alır; ilk dört parametre başlangıç ve bitiş koordinatlarını belirtirken, beşinci parametre ise kaydırma hızını ayarlamak için kullanılır. *Menü* ve *Geri* eylemleri mobil cihaz üzerindeki ilgili düğmelerin basılmasını temsil eden eylemlerdirler ve herhangi bir parametre almazlar. Bağlamsal eylemler, kullanıcının Test Altındaki Uygulamanın (TAU) bağlamsal durumunu değiştirdiği eylemleri ifade eder. Mobil cihazın global niteliklerinin birleşimi (internet bağlanırlığı, bluetooth durumu, konum, uçak modu ve uyku modu) uygulamanın o anki bağlamsal durumunu oluşturur. *Bağlanırlık* eylemi mobil cihazın internet bağlanırlığını ayarlar (Wi-Fi veya mobil veri). *Bluetooth durumu*, *konum* ve *uçuş modu* nitelikleri açık ve anlaşılırdır. *Uyku* eylemi mobil cihazı güç düğmesine basarak uyku moduna alan veya uyku modundan çıkaran eylemdir. *Uyku* eylemi test edilen uygulamayı duraklatmak ve devam ettirmek için kullanılır. Özel (special) eylem olarak da uygulamayı yeniden yükleyip başlatmaya yarayan *yenidenbaşlatmak*

Tablo 1. GKA Eylemler Listesi

Bağlamsal olmayan	Param1	Param2	Param3	Param4	Param5
tıklama	x	y	-	-	-
uzuntıklama	x	y	-	-	-
metin	x	y	string	-	-
kaydırma	x1	y1	x2	y2	süre
menü	-	-	-	-	-
geri	-	-	-	-	-
Bağlamsal	Parametre				
bağlanırlık	açık/kapalı/değiştir				
bluetooth	açık/kapalı/değştir				
konum	gps/gps&ağ/kapalı/değiştir				
uçuşmodu	açık/kapalı/değiştir				
uyku	açık/kapalı/değiştir				
Özel	Param1	Param2	Param3	Param4	Param5
yenidenbaşlatmak	paket	aktivite	-	-	-

(reinitialize) bulunmaktadır. Sistem olayları sistem tarafından oluşturulan olaylardır; örneğin, *pil seviyesi* olayları, *SMS almak*, ve *saat/süreölçer* olayları gibi.

Son olarak, bir çökmeyi Android sistem kayıtlarında görülen bir ölümcül hata (fatal exception) olarak tanımlıyoruz. Çökmeler, sıklıkla test edilen uygulamanın bir uyarıyla veya uyarısız bir şekilde sonlanmasıyla sonuçlanır. Bazı çökmeler ise yürütmeyi görsel olarak etkilemez, fakat test edilen uygulama sonuç olarak çalışmayı durdurur.

Bir GKA durumu veya kısaca bir *durum* v aşağıdaki ögelerin bitiştirilmesinden (concatenation) oluşur.

1. Paket adı
2. Aktivite adı
3. Bağlamsal durum
4. GKA bileşenleri

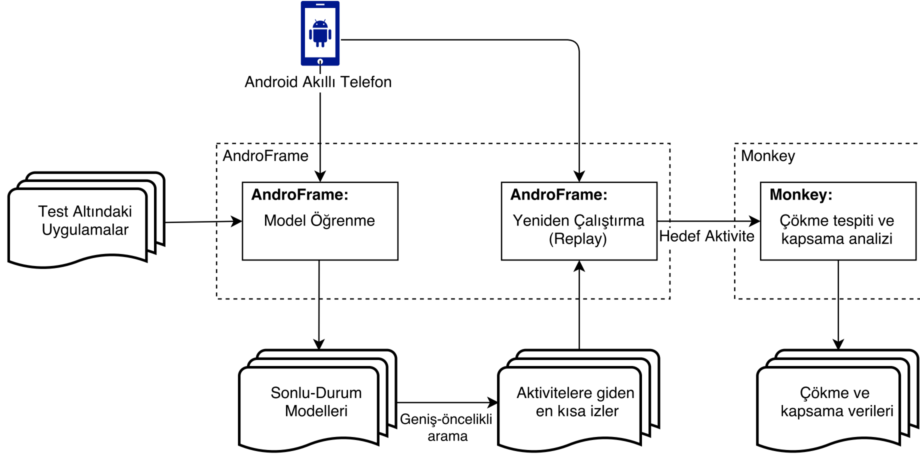
Her durum v için GKA bileşenlerinden elde edilebilen bir etkin eylemler kümesi $\lambda(v)$ vardır. Bir GKA eylemi veya kısaca *eylem* $z \in Z$, ancak ve ancak bir v durumunun GKA bileşenlerinden en az biri ile ilişkilendirilebiliyorsa, z eylemi v durumunda etkindir, kısaca $z \in \lambda(v)$, denilir. Bir *geçiş*, (başlangıç-durumu, bitiş-durumu, eylem) olacak şekilde üçlü değişkenler grubu (tuple) olarak tanımlanır. Bir yürütme izi (execution trace) veya kısaca *iz (trace)* t, bir geçişler dizisidir. Örneğin n uzunluğa sahip bir iz aşağıdaki gibi olabilir.

$$t = (v_1, v_2, z_1), (v_2, v_3, z_2), \ldots, (v_n, v_{n+1}, z_n)$$

Eğer bir iz t'nin ilk durumu, TAU başlatıldığı andaki GKA durumu olan ilk durum v_0 ile aynıysa, t bir *test örneği*dir (test case). Sınama örneklerini içeren kümelere *test kümesi* (test suite), kısaca TK denilir.

3 Yöntem

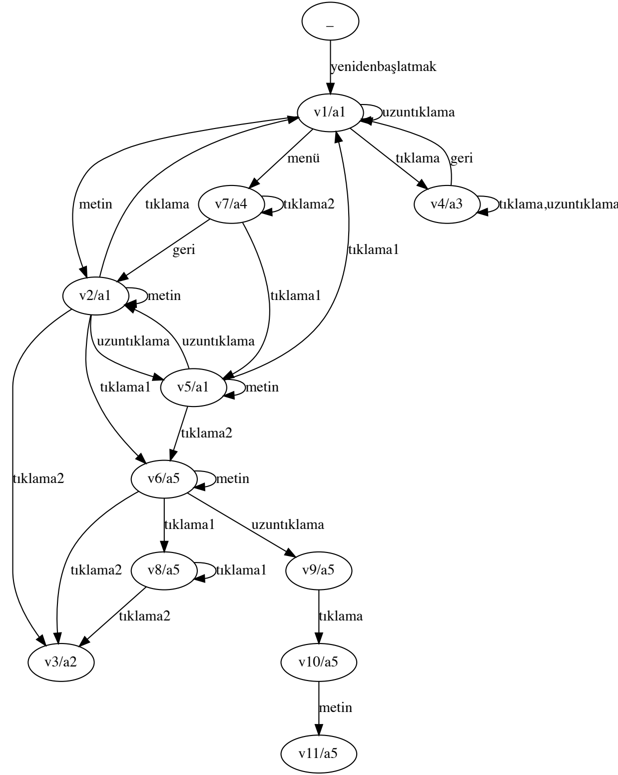
Bu bölümde AFM aracının akış çizelgesi açıklanmaktadır. Akış çizelgesinin daha iyi anlaşılması açısından bir örnek üzerinden detaylı açıklamalar da bulunmaktadır.



Şekil 1. AFM Aracının Akış Çizelgesi (Flowchart)

Şekil 1, AFM aracının akış çizelgesini göstermektedir. İlk olarak AndroFrame'in model öğrenme altyapısını kullanarak Test Altındaki Uygulamaların (TAU) sonlu-durum modelleri çıkarılmaktadır. Bu modeller üzerinden genişlik-öncelikli arama (breadth-first search) yöntemi ile her uygulamanın her farklı aktivitesine giden en kısa izler belirlenir. Daha sonra bu izler AndroFrame'in yeniden çalıştırma (replay) özelliği kullanılarak mobil cihaz üzerinde birer birer çalıştırılır. Her izin çalıştırılmasından sonra, belli bir süre boyunca Monkey çalıştırılarak çökme tespiti ve kapsama analizi yapılır. Monkey'in her aktivite için çalıştırılma süresi, TAU modeli için çıkarılmış izlerin sayısı ile ters orantılıdır. Diğer test araçlarıyla adil bir kıyaslama yapılabilmesi adına AFM'nin çalışma süresi sabit tutulmuş (10 dakika), her iz sonunda Monkey çalıştırma süresi de bu sabit sürenin iz sayısına bölünmesi ile hesaplanmıştır.

Şekil 2, internetteki açık kodlu F-Droid [6] test uygulamalarından AndroidomaticKeyer uygulamasının öğrenilmiş bir sonlu-durum modelini göstermektedir. *Yenidenbaşlatmak* eylemi mobil cihazın herhangi bir durumunda yapılabilmekte olduğundan bu eylemden önceki durum içeriği önemsiz anlamında '_' ile gösterilmiştir. Diğer her durumda, $v\#$ o duruma verilmiş özel ismi, $a\#$ ise o durumun ait olduğu aktiviteyi gösterir. Sonlu-durum makinesinde her aktivite için birden fazla durum bulunabilir. Her iki durum arasındaki geçişlerdeki eylemler kısaltılarak sadece olayların tipleri belirtilmiştir.

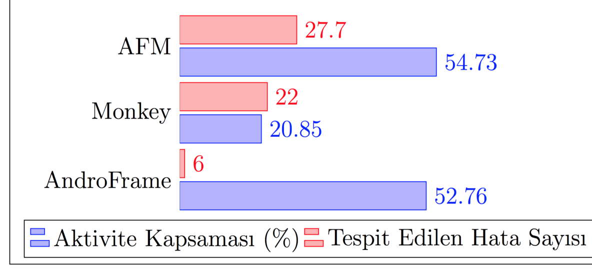


Şekil 2. Örnek TAU Sonlu-Durum Modeli

Tablo 2. Çalıştırılacak En Kısa İzler Listesi

Aktivite	**Aktiviteye Ulaşan En Kısa İz**
a1	t_{a1} = (_,v1,*yenidenbaşlatmak*)
a2	t_{a2} = (_,v1,*yenidenbaşlatmak*), (v1,v2,*metin*), (v2,v3,*tıklama2*)
a3	t_{a3} = (_,v1,*yenidenbaşlatmak*), (v1,v4,*tıklama*)
a4	t_{a4} = (_,v1,*yenidenbaşlatmak*), (v1,v7,*menü*)
a5	t_{a5} = (_,v1,*yenidenbaşlatmak*), (v1,v2,*metin*), (v2,v6,*tıklama1*)

Tablo 2'de, Şekil 2 üzerinde genişlik-öncelikli arama sonucu bulunmuş en kısa izler gösterilmektedir. Bu izlerin her biri uygulamanın 5 farklı aktivitesinden birine gitmektedir. AFM 10 dakika çalıştırılacağından, her izin çalıştırılıp ardından Monkey kullanılması için ikişer dakika ayrılmıştır. Monkey'in sadece bir kere 10 dakikalığına ilk durumdan çağrılmasına kıyasla AFM yöntemi, uygulamanın sonlu-durum modelindeki bütün aktiviteleri kapsamayı garanti etmektedir. Ayrıca Monkey'in tek başına çağrıldığında $v3$ durumundan geri dönüş yapmak



Şekil 3. AndroFrame, Monkey, ve AFM Deney Sonuçları

mümkün olmazken, AFM yönteminde iki dakikanın bitiminde hemen başka bir izin çalıştırılmasına geçilir.

4 Deneyler

Bu bölümde AFM'nin kıyaslanması için gerçekleştirdiğimiz deneylerin içeriği ve sonuçları açıklanmaktadır.

Deneylerin çalıştırılması için Android Debugging Bridge (adb) aracının en yeni versiyonu kurulmuştur. Deneyler emulasyon ortamında, Android 4.4.2 versiyonu üzerinde koşulmuştur.

Adil bir kıyaslama yapabilmek adına AndroFrame, Monkey, ve AFM araçlarının her biri her TAU üzerinde 10 dakika çalıştırılmıştır. Çökme tespiti ve kapsam analizi kıyaslamaları için 20 adet Android uygulaması F-Droid [6] sitesinden indirilmiştir. Kıyaslama ölçütü olarak her uygulamanın aktivite kapsaması ve çökme tespit sayıları AndroFrame, Monkey, ve AFM için hesaplanmıştır.

Android GKA Test deneylerinde başarım kriteri olarak ortalama aktivite kapsaması ve çökme tespit sayılarına bakmaktayız. Deneylerimizin rastgelelik içermesinden ötürü tüm deneylerimizi üçer kere tekrarlayıp sonuçlarımızın tekrar ortalamasını Şekil 3 ile raporlamaktayız. AFM, Monkey'e ve AndroFrame'e kıyasla daha fazla hata tespitinde bulunmaktadır. Bunun temel nedeni AFM'nin derinliklerdeki aktivitelerde de çok hata tespiti yapabilen Monkey aracını çalıştırabilmesidir. AFM'nin aktivite kapsaması yine Monkey ve AndroFrame'e göre daha iyi çıkmıştır. AFM, algoritması gereği AndroFrame'in ulaştığı tüm aktivitelere ulaşmaktadır. Bu aktivitelerde Monkey çalıştırılması sonucu daha önce keşfedilmemiş yeni aktivitelere gidilmiştir. Böylece AFM'nin aktivite kapsaması AndroFrame aracına göre daha yüksek çıkmıştır. Sonuç olarak, AFM, 20 adet Android uygulaması içerisinde AndroFrame'e kıyasla %2 daha fazla aktivite kapsamasına ulaşıp ortalamada fazladan yaklaşık 21 çökme tespit etmiştir. Monkey'e kıyasla ise %24 daha fazla aktivite kapsamasına ulaşıp ortalamada fazladan yaklaşık 5 çökme tespit etmiştir.

5 Tartışma

5.1 Monkey Hakkında

Monkey sayıca diğer test araçlarına göre daha çok sayıda çökme tespit eden basit bir rastgele test yaratım aracıdır [10]. Monkey'in başarısının temelinde diğer test araçları tarafından üretilemeyen olaylar bulunmaktadır. Bu olaylara örnek olarak birden fazla parmakla yapılabilen karışık hareketler, aktivitelere yeniden başlatma dışında intentler yollamak, trackball hareketleri, ve yazılımda tanımlı ama donanımda bulunmayan özel tuşlara basmak verilebilir. Monkey bu olayları Android işletim sistemine gömülü olarak geldiği için yapabilmektedir. Diğer test araçları genel olarak Android işletim sistemine dışarıdan *uiautomator* veya daha eski *troyd* altyapılarını kullanarak Android cihaza erişirler. Bu altyapılar söz konusu olayları desteklememektedir.

Monkey AndroFrame'e göre daha az aktivite kapsamasına ulaşabilmektedir. Bunun nedeni Şekil 2 ile verilen modelden kolayca anlaşılabilir. Monkey, çok sayıda çökme tespit etmesine rağmen geri dönüşü mümkün olmayan durumlara takılı kaldığı ya da sayıca çok olay arasından aktivite geçişi için gerekli olanı tutturamadığı için, TAU derinliklerinde kalan başka aktivitelere erişememektedir. AFM'yi, Monkey aracının bu aktivitelerde de çalıştırılmasının çökme tespit sayısını artıracağı fikrinden yola çıkarak önermekteyiz.

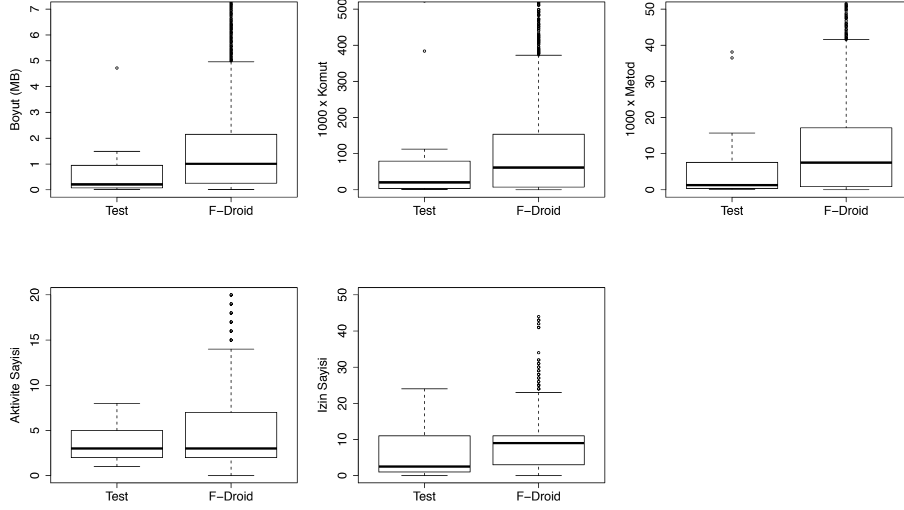
Monkey hakkındaki bir diğer sorun ise Monkey ile üretilen testlerin kolayca yeniden çalıştırılabilir betikler (replayable script) haline getirilememesidir [11]. Bu problem AFM'nin tespit ettiği çökmelerin doğrulanamamasına yol açmaktadır. İleride Monkey ile üretilen testlerin tekrar edilebilmesi amacıyla AFM aracımızı geliştirmeyi planlamaktayız.

Monkey çalıştırırken yaratılan bir testte çökmeye hangi olayın neden olduğuna karar vermek açık bir problemdir [11]. Şu an için AFM, bir iz çalıştırdıktan sonra Monkey'i birden çok eylem gerçekleştirecek şekilde çalıştırmaktadır. AFM, Monkey'i her bir eylem için çalıştırıp/durdurarak da Monkey testi yapabilir. Böylece AFM çökmeye hangi eylemin sebep olduğunu ortaya çıkarabilir. Bu yöntemi, Monkey'in kısıtlı sürede ürettiği eylem sayısını azaltıp çökme tespit sayısını düşürebilme ihtimali yüzünden bu çalışmamızda tercih etmemiş bulunmaktayız.

5.2 AndroFrame Hakkında

AndroFrame modüler bir test yaratım aracıdır. AndroFrame içerisinde derinlik-öncelikli, rastgele, ve makine-öğrenmesi tabanlı arama stratejileri kodlanmıştır. Bu çalışmamızda AndroFrame makine-öğrenmesi tabanlı arama yöntemiyle çalıştırılmıştır. AndroFrame içerisindeki makine-öğrenme modülü aktivite kapsamasını artıracak olan olaylara öncelik verecek şekilde eğitilmiştir. Bu sayede AndroFrame, Monkey'e göre daha yüksek aktivite kapsamasına ulaşmaktadır.

AndroFrame'in öğrendiği sonlu-durum modeli tam değildir. Dolayısıyla öğrenilmiş modellerden çıkarılan izlerin istenilen aktivitelere gittiklerini doğrulayıp,



Şekil 4. Test ve F-Droid Uygulama Karakteristiklerinin Dağılımları

eğer istenilen aktiviteye ulaşılamamış ise model öğrenirken kullanılan test örneklerinden o aktiviteye ulaştığı bilinen izlerden birini seçmek ilerisi için pratik bir çözüm olabilir.

5.3 Geçerlilik Sorunları

Deneylerimizde emulasyon ortamından yararlanmaktayız. Emulasyon ortamı şarj bitmesi gibi sistem olaylarını doğru yansıtmayabilir. Deneylerimizde kullandığımız eylemlerin hepsi emulasyon ortamında desteklenmektedir. Ayrıca tüm araçların aynı koşullar altında çalıştırılması deneylerimizin kuvvetini artırmaktadır.

F-Droid uygulamaları [6] AFM test aracının diğer araçlarla kıyaslanması için yeterlidir, çünkü birçok üst seviye konferans bildirisi ve dergi makalesinde Android test araçlarının kıyaslanması için kullanılmıştır [4,5,9,11].

F-Droid uygulamaları Mart 2016 ayı itibariyle 4021 adettir ve bu sayı gün geçtikçe artmaktadır. İndirdiğimiz 4021 uygulamadan deneylerimiz için 20 taneyi rastgele seçmiş bulunmaktayız. Seçtiğimiz uygulamaların F-Droid uygulamalarının genelini temsil ettiğini gösterebilmek adına bu uygulamaların komut, metod, aktivite, ve izin sayıları ile megabayt cinsinden boyutlarını ölçtük. Şekil 4 ve Tablo 3, test ettiğimiz 20 uygulamanın (test kümesi) ve tüm F-Droid uygulamalarının karakteristiklerini göstermektedir.

Aktivite kapsaması ölçümü, Android testlerinde yaygın olarak kullanılmaktadır [1]. Aktiviteler, kabaca geleneksel GKA-tabanlı uygulamalardaki farklı ekranlara ya da pencerelere karşılık gelirler. Aktivite kapsamasını artırmak, daha

Tablo 3. Test ve F-Droid Uygulamaların Asgari, Azami, ve Ortalama Karakteristikleri

	Test Kümesi			F-Droid		
Karakteristik	Asgari	Azami	Ortalama	Asgari	Azami	Ortalama
Boyut (MB)	0.02	17.48	1.8	0.01	157.2	2.29
1000 x Komutlar	0.9	522.2	74.17	0.01	1395	107.4
1000 x Metodlar	0.18	38.15	6.51	0.01	157.7	11.3
# Aktiviteler	1	28	6.3	0	123	5.79
# İzinler	0	24	6.4	0	168	8.4

fazla ekranı keşfetme anlamına gelmektedir. Daha fazla ekranı keşfetmek ise, uygulamanın daha çok işlevini kapsamaktadır. Bu yüzden, yaratılan test örneklerinin TAU'nun daha çok işlevini kapsaması için daha yüksek aktivite kapsamasını hedeflemekteyiz. Aktivite kapsamasının avantajı, uygulamanın kaynak kodunun değiştirilmesine (instrumentation) gerek duymamasıdır. Komut kapsaması gibi diğer yaygın olarak kullanılan ölçümleri yapabilmek için TAU'nun kaynak kodunun değiştirilmesi gerekmektedir.

Çökme sayıları ve kapsama ölçümleri, yalnızca deney ortamımızda kullanılan test etme algoritmasından etkilendiği için, gözlemlerimizin içsel geçerliliği (internal validity) korunmaktadır. Test araçlarının adil bir karşılaştırmasını yapabilmek adına, çökme ve aktivite kapsamasını bütün araçlarda aynı yöntemleri kullanarak ölçmekteyiz. Doğruluğu artırmak amacıyla, diğer test araçları da ortalama olarak benzer bir hızla olay oluşturduğu için, Monkey'i de iki saniyede bir olay oluşturacak şekilde çalışmaya zorladık. F-Droid uygulamaları, Android GKA testi çalışmalarında yaygın olarak kullanılmaktadır [5]; bu yüzden test kümemizin seçim yanlılığına (selection bias) eğilimi yoktur.

Gözlemlerimiz dışsal geçerliliğini (external validity) de korumaktadır. Kullandığımız uygulamaları, F-Droid uygulama kümesinden rastgele olarak indirdik. Kullandığımız uygulamaların rastgeleliği, çeşitliliği ve sayısı, bu uygulamalar üzerindeki deney sonuçlarının dışsal olarak genellenebilir olduğunu göstermektedir. Uygulama kümemiz, haber, eğlence ve iletişim defteri uygulamaları gibi çeşitli alanlardaki uygulamalardan oluştuğu için kapsamlıdır. Ayrıca kullandığımız TAU'ların ortalama karakteristik özelliklerini de Şekil 4 ile göstermekteyiz. Bu karakteristiklere sahip olduğu sürece, verilen bütün TAU'lar için benzer sonuçları almayı beklemekteyiz.

6 İlgili Çalışmalar

AndroFrame test aracını, Test Altındaki Uygulamaların (TAU) sonlu-durum modellerini öğrenmek için kullanmaktayız. AndroFrame yetenekleri itibariyle aşağıda anlatılan son model (state-of-the-art) test araçlarıyla karşılaştırılabilir durumdadır.

A^3E [1] sistematik olarak GKA bileşenlerini çalıştıran bir Android test aracıdır. A^3E, uygulamanın birden fazla olabilen giriş noktalarını (exportable activities) desteklemektedir. AndroFrame de bu aktiviteleri desteklemektedir.

CrashScope [11], *wifi* ve *rotation* gibi bağlamsal eylemleri ortaya koymaktadır. CrashScope bağlamsal durumları değiştirmenin yeni çökmeleri ortaya çıkardığını göstermektedir. AFM, *döndürme* eylemi dışındaki bütün bağlamsal eylemleri desteklemektedir. *Döndürme* desteğini ileriki bir çalışma olarak kodlayacağız.

SwiftHand [4] TAU'nın sonlu-durum modelini yaratmak için özdevinir öğrenme (automata learning) kullanmaktadır. AndroFrame'de algoritmalarımızı tanımlarken genel olarak SwiftHand'in biçimselleştirmesini takip etmekteyiz. Ayrıca, TAU'nın deterministik bir modelini elde etmek için SwiftHand'in PassiveLearn algoritmasını kullanmaktayız.

PUMA [7] bir başka kara-kutu Android test aracıdır. PUMA'nın ana katkısı, iki durumun içeriklerinin karşılaştırılmasına dayanan *kosinüs benzerliği*dir. AndroFrame kosinüs benzerliğini durum eşdeğerliği için kullanmaktadır.

Baek and Bae [2], Android GKA durumları için bir karşılaştırma kriteri tanımlamaktadır. AndroFrame, bu çalışmada anlatılan maksimum karşılaştırma seviyesini kullanarak kara-kutu testi için modellerimizi olabildiğince ince-taneli (fine-grained) yapmaktadır.

7 Sonuç

Bu çalışmamızda model öğrenen AndroFrame aracımızı Monkey ile karma olarak çalıştırarak daha çok aktiviteyi kapsayıp çökme tespit performansını iyileştirdik. Karma yöntemimiz 20 adet Android uygulaması içerisinde AndroFrame'e kıyasla %2 daha fazla aktivite kapsamasına ulaşıp fazladan 21 çökme tespit etmiştir. Monkey'e kıyasla ise %24 daha fazla aktivite kapsamasına ulaşıp fazladan 5 çökme tespit etmiştir.

İlerideki çalışmalarımızda deney sayısını artırmayı, diğer test araçları ile kıyaslama yapmayı, ve deneyleri AndroFrame'in diğer arama stratejileri için de uygulamayı planlıyoruz. Ayrıca, Monkey ile üretilen testlerin yeniden çalıştırılabilir hale getirilip testlerin yeniden çalıştırılarak çökme tespitlerinin doğrulanması da başlıca izleyeceğimiz araştırma yolları arasındadır.

Kaynaklar

1. Azim, T., Neamtiu, I.: Targeted and depth-first exploration for systematic testing of android apps. In: Proceedings of the ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA) (2013)
2. Baek, Y.M., Bae, D.H.: Automated model-based android gui testing using multilevel gui comparison criteria. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE) (2016)
3. Chaffey, D.: Statistics on consumer mobile usage and adoption to inform your mobile marketing strategy mobile site design and app development (2017), http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/

4. Choi, W., Necula, G., Sen, K.: Guided gui testing of android apps with minimal restart and approximate learning. In: Proceedings of the ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA) (2013)
5. Choudhary, S.R., Gorla, A., Orso, A.: Automated test input generation for android: Are we there yet? In: Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering. ASE (2015)
6. Gultnieks, C.: F-Droid Benchmarks (2010), https://f-droid.org/
7. Hao, S., Liu, B., Nath, S., Halfond, W.G., Govindan, R.: Puma: Programmable ui-automation for large-scale dynamic analysis of mobile apps. In: Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys) (2014)
8. Koroglu, Y., Sen, A.: AndroFrame Technical Report (2017), https://www.cmpe.boun.edu.tr/ yavuz.koroglu/AndroFrameTechnicalReport.pdf
9. Machiry, A., Tahiliani, R., Naik, M.: Dynodroid: An input generation system for android apps. In: Proceedings of the 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE) (2013)
10. Android ui/application exerciser monkey, http://developer.android.com/tools/help/monkey.html
11. Moran, K., Vásquez, M.L., Bernal-Cárdenas, C., Vendome, C., Poshyvanyk, D.: Automatically discovering, reporting and reproducing android application crashes. In: IEEE International Conference on Software Testing, Verification and Validation (ICST) (2016)
12. Zein, S., Salleh, N., Grundy, J.: A systematic mapping study of mobile application testing techniques. J. Syst. Softw. 117, 334–356 (2016)