Hibrid Algoritma ve Isıl İşlem Algoritmasıyla Test Kümesi Önceliklendirilmesi

Şefik Temel[1], M. Özgür Cingiz[2], Oya Kalıpsız[3]

[1,2,3]Bilgisayar Mühendisliği Bölümü, Yıldız Teknik Üniversitesi, İstanbul

[1] sefiktemel@gmail.com
[2,3] { mozgur@ce.yildiz.edu.tr , kalipsiz@yildiz.edu.tr}

Özet. Yazılımlar yaşam süresince müşteri istekleri, gereksinimler ve tasarım kararlarına göre değişiklik gösteren dinamik bir yapıdadır. İlgili bu güncellemelerin uygulamaya etkileri gözetilerek değişikliklerin kontrol edilmesi gerekilmektedir. Güncellenen yazılımlarda değişen modüllerin tekrar testlerinin yapılması bakım aşamasının en maliyetli süreçlerinden biridir. Bu sürecin bir parçası olan regresyon testinde testler önceliklendirilerek en hızlı şekilde hataların tespiti sağlanıp bakım sürecinin maliyeti azaltılmaktadır. Çalışmamızda 1000 farklı hata ve bu hataları tespit eden 100 farklı testin, testlerin hataları belirleme oranına göre sıralanması hibrid algoritma ve ısıl işlem algoritmasıyla gerçeklenmiş ve bu algoritmaların sonuçları rassal sıralama yöntemiyle karşılaştırılarak ciddi bir performans arttırımı gerçeklendiği gözlemlenmiştir.

Anahtar Kelimeler: Test kümesi önceliklendirme, Regresyon testi, Yazılım test mühendisliği, Yazılım tekrar kullanım

1. Giriş

Yazılım geliştirme süreci durağan olmaktan öte aktif bir süreçtir. Bu süreçte müşteri odaklı veya ihtiyaç odaklı modüler değişikler gerçekleşebilmektedir. Yapılan bu değişiklerin kontrolü ve testinin yapılması çok önemlidir. Yazılım geliştirme sürecinde en maliyetli süreç bakım ve onarım kısmıdır. Bakım ve onarım sürecinde tasarımın düzgün yapılamaması, süreçlerin düzgün tanımlanamaması, yanlış yazılım mimarisi seçimi, değişiklik kontrolünün yapılmaması gibi pek çok etmen bulunmaktadır.

Uygulamada ve uygulama ortamlarında gerekli değişiklikler, testler ve sabitlemeler yapıldıktan sonra kontrol için yeniden yapılan testlere regresyon testi denilir. Regresyon testiyle önceki testlerde belirlenen sorunların giderildiğinden ve değişiklikler sonucunda yeni hatalar oluşup oluşmadığı incelenir. Uygulamanın kaç kez yeniden test edilmesi gerektiğini belirlemek güçtür ve bu nedenle özellikle uygulama geliştirme döneminin sonlarına doğru yani bakım aşamasında daha sıklıkla regresyon testleri yapılmaktadır. Bakım aşaması tüm yazılım geliştirme sürecindeki

tüm maliyetin %40 ile %80 arasına değişen bir yüzdeye denk geldiği için [1] regresyon testlerinin önemi ortaya çıkmaktadır.

Rothermel ve arkadaşları [2] regresyon testini, değişiklik yapılmış yazılımda yeni hataların ortaya çıkıp çıkmadığının ve değişiklerin uygunluğunun kontrolü olarak tanımlamıştır. Regresyon testlerinin uygulanması, regresyon testinin hangi sıklıkla tekrarlanması gerektiği aynı zamanda problematiktir. Yazılımdaki her bir değişiklikten sonra yazılımın tüm fonksiyonel ve fonksiyonel olmayan gereksinimleri karşılayıp karşılamadığını kontrol etmek anlamlı olmayabilir. Bu açıdan yapılan değişikliklerin kontrolünün nasıl ve ne sıklıkla gerçekleştirilmesi gerektiğini belirlemek önemli bir iştir. Harrold var olan testlerle ve yazılımda değişiklik yapıldığında bu değişikliklerden etkilenecek bölümlerin bulunmasıyla regresyon testinin maliyetinin azaltılacağını belirtmiştir [2,3]. Son yıllarda en az sayıda test ile hata bulma oranını arttırmaya yönelik pek çok çalışma yapılmıştır. Bu çalışmalar aynı zamanda regresyon testinin maliyetini düşürmeyi hedeflemektedir.

Regresyon test seçme teknikleri, tüm yazılım testleri içerisinden yazılımın en son güncellenmiş sürümündeki hataları, değişiklikleri en iyi şekilde belirleyen alt bir test kümesi belirlemeye çalışmaktadır [4,5]. Test kümesi azaltma teknikleri [6] ise test seçme tekniğinden farklı olarak en iyi alt küme yerine hata ve değişiklik takibi yapamayan testlerin elenmesi işlemini gerçekleştirmektedir. Test seçme ve test kümesi azaltma teknikleri en iyi hataları keşfeden alt kümeler üzerinde çalışırlar, bu nedenle test kümesinin hata bulma yüzdesini düşürebildikleri gözlemlenmiştir [7,8]. İlgili tekniklerdeki problemler yüzünden Yazılım Test Önceliklendirme (YTÖ) teknikleri araştırmacıların ilgi alanı haline gelmiştir. Yazılım test önceliklendirmeyle test kümesindeki testler belirli bir sırada uygulanırlar. Önceliklendirmedeki amaç test kümesinde yer alan çok sayıda test içerisinde en son değiştirilmiş yazılımdaki hataları en yüksek oranda belirleyen testlerin öncelikli olarak belirlenerek ilk bu testlerin uygulanmasıdır. Böylece test süresince hızlı bir şekilde hata keşfi yapılmaktadır [9].

Son yıllarda bu alanla ilgili yapılan çalışmalarda özellikle meta-sezgisel yöntemler özellikle daha çok tercih edilmektedir [10]. Bu yöntemler kullanılarak yazılımda yer alan hataları en yüksek oranda belirleyen testler önceden uygulanarak hataların erken keşfi sağlanmaktadır.

Çalışmamızda bir program parçasında yer alan 1000 farklı hata ve bu program parçasındaki hataları tespit eden 100 farklı test kümesi rassal olarak oluşturulmuştur. Her bir hata birden çok test tarafından veya sadece tek bir test tarafından kontrol edilebilmektedir. Her bir testin hangi hataları kontrol ettiği önceden bilindiği düşünülmekte ve böylece optimal bir test kümesi ile çalışılmaktadır.

Çalışmamız kapsamında hangi hataları tespit edebildiği önceden belirli olan testler rassal olarak tanımlanmıştır. Bir hatayı belirleyen birden fazla testin olabileceği göz önüne alınmıştır. İlgili hataları test eden testler iyileştirilmiş genetik algoritma ve ısıl işlem algoritması yöntemleri kullanılarak önceliklendirilmiştir. Böylece hataları test eden testlerin ilk olarak test aşamasından uygulanmasıyla regresyon testi süresinin azaltılması ve az sayıda test ile ilgili hataların tespit edilmesi amaçlanmaktadır.

Çalışmamızda bölüm 2'de sistem tasarımına, bölüm 3'te ise deneysel sonuçlara ve değerlendirmelere yer verilmiştir.

2. Sistem Tasarımı

Bu bölümde farklı sayıdaki hataları belirleyen testlerin iyileştirilmiş hibrit algoritma ve ısıl işlem algoritması ile ne şekilde önceliklendirildiği üzerinde durulmuştur.
Yaptığımız çalışmada, bir yazılımda 1000 farklı hata olduğu ve bu hataların tamamını belirleyebilen 100 farklı test olduğu varsayılmıştır. Her hatanın önemliliği farklı olacağından bu hatalara 1 ile 10 arasında önem değerleri verilmiştir. Hangi hatanın hangi test tarafından belirlendiği ise rassal olarak belirlenmiştir. Tablo 2.1'de hatalar ve bu hataları belirleyen testler örnek olarak verilmiştir.

Tablo 2.1 Hatalar ve Hataları Tespit Eden Testler

Test	**Hatalar**								
	1	2	3	…	99	100	…	999	1000
T1	x	x	x					x	
T2	x				x				X
T3			x			x		x	
⋮									
T100		x			x				X

Yaptığımız çalışmada elde ettiğimiz önceliklendirmede tüm test kümesi uygulanmaktadır, fakat önemlilik derecesi yüksek testler daha önce uygulanarak daha kısa zamanda önemli hatalardan daha fazlasının tespit edilmesi sağlanmaktadır. Yazılım test önceliklendirmesi için uygulanan iyileştirilmiş genetik (hibrid) algoritmanın işleyişi Şekil 2.1'de verilmiştir.

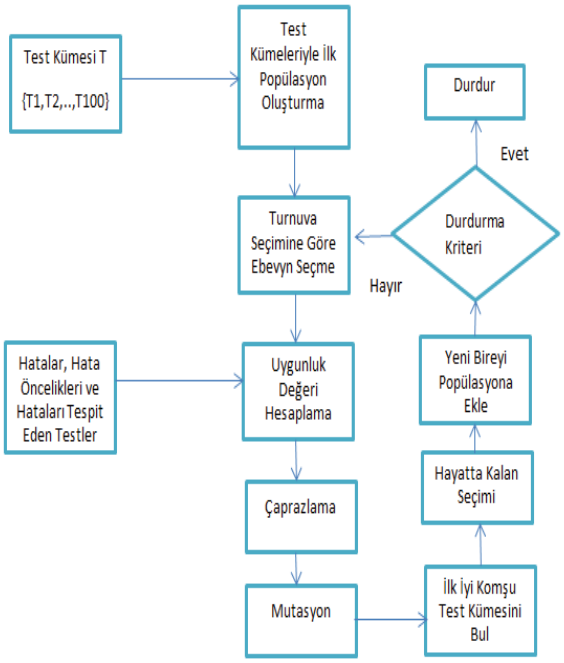
Yazılım test önceliklendirmedeki ana fikir testleri hata bulma oranlarına göre sıralanmaktır. Şekil 2.1'de görüldüğü üzere iyileştirilmiş genetik algoritmanın ilk adımı ilk popülasyonun oluşturulmasıdır. Bu popülasyon rassal olarak belirlenmiştir. İyileştirilmiş genetik algoritma kullanılarak en iyi durum bu popülasyondan üretilmiştir.

İyileştirilmiş genetik algoritmanın popülasyon modeli kararlı durumdur (Steady State). Kararlı duruma göre popülasyondaki birey sayısı her bir jenerasyonda sabittir. Çalışmamızda popülasyondaki birey sayısı 50 olarak belirlenmiştir ve permütasyon gösteriminden yararlanılmıştır.

Tablo 2.2 İlk Popülasyon Örneği

Bireyler	**Testlerin Uygulanış Sırası**
Test Kümesi 1	T47, T14, T61, …,T11, T34, T7,..,T91,T3
Test Kümesi 2	T9, T21, T1, …,T88, T54, T91,..,T74,T32
. . .	
Test Kümesi 50	T33, T65, T52, …,T8, T4, T17,..,T94,T25

Tablo 2.2'de gösterildiği gibi 50 farklı test kümesi ve bu test kümesindeki testler rassal öncelik değerlerine göre belirlenerek ilk popülasyon 50 birey ile (50 farklı test kümesiyle) oluşturulmuştur.



Şekil 2.1 İyileştirilmiş Genetik Algoritma İle Yazılım Test Önceliklendirme Adımları

Tablo 2.2'de örnek olarak verilmiş ilk birey yani ilk test kümesi incelendiğinde 100 test içerisinden ilk uygulanan testin 47'inci test olduğu, bu testi 14'üncü testin takip ettiği ve en son uygulanan testin ise 3'üncü test olduğu gözükmektedir.

Şekil 2.1'deki sistem tasarımımıza göre ilk popülasyon oluşturulduktan sonra turnuva seçim yöntemine göre ebeveyn seçimi yapılmaktadır. Popülasyondaki 50 birey arasından rassal olarak seçilen 5 bireyin içerisinden en uygun birey ilk ebeveyn, aynı işlem tekrarlanarak seçilen ikinci birey ise ikinci ebeveyn olarak belirlenmektedir.

Tablo 2.3 Testler ve Testlerin Belirlediği Hatalar

Testler	**Testlerin Belirlediği Hatalar**
T1	H13, H743, H530,H 403, H21,H4
T2	H6,H134,H318,H489,H614,H777,H910
⋮	
T100	H11,H109,H222,H578,H729

Genetik algoritmalarda uygunluk fonksiyonunun (Fitness Function) doğru olarak belirlenmesi çok önemlidir. Çalışmamızda bireylerin uygunluğu (test kümesindeki testlerin sıralanışlarının uygunluğu), önceliklendirme yapılan testlerin belirlediği hata sayısına, testin uygulanış sırasına ve hatanın öncelik değerine bağlıdır.

Tablo 2.3'te rassal olarak hatalar ve bu hataları belirleyen testler birlikte gösterilmiştir. Buna göre örneğin test 1 13'üncü hata, 743'üncü hata gibi yazılım hatalarını belirleyebilmektedir. Yazılımdaki her bir hatanın önemi farklı olabileceği için hataların önem sıraları yine rassal olarak 1 ile 10 arası ağırlıklandırılmıştır. Bun göre uygunluk fonksiyonu popülasyondaki bireyler için aşağıdaki şekilde tanımlanmaktadır.

$$Uygunluk\ Değeri = \sum_{i=0}^{99} (100 - i) \sum_{k=1} hata\ önem_k \qquad (1)$$

Şekil 2.2 Uygunluk Fonksiyonu

Şekil 2.2'de verilen denkleme göre test kümesinde ilk sırada olan yani en öncelikli olarak belirlenen testin alacağı ilk test sırası katsayısı 100 olacaktır, ikinci testin 99, sonuncu testin ise 1 olacaktır. Uygunluk fonksiyonunun ikinci çarpanı ise her bir testin belirlediği hata ve bu hataların önem katsayılarını kullanarak teste ait hata belirleme değeridir. Uygunluk fonksiyonunun nasıl hesaplandığına dair küçük bir örnek vermek gerekirse:

- Farklı şekilde dizilmiş ve 1000 hata içerisinden rassal olarak oluşmuş 100 hatanın testler tarafından bulunmasının istendiğini düşünelim ve testlerin sırayla test 47, test 13, test 43… test 55 olarak dizilsin. (Uygulama sırasına göre 100 tane testin dizilişi)
- Test 47'nin, 100 test arasında ilk uygulanan test olmasından dolayı ilk çarpan 100 olacaktır.
- Test 47'nin 100 hata içerisinden belirlediği hata sayısı 3 olsun ve hataların önem sıraları 1-10 arası değerler olan 4,1,9 olarak verilsin. İkinci çarpan olarak elde edeceğimiz değer bu önem değerlerinin toplamı olan 14'tür. Bu nedenle ilk test

47 olarak belirlendiğinde 47'inci testten elde edilen uygunluk değer 14*100=1400'dür. Bunun gibi tüm 100 test için uygunluk değeri hesaplanarak önem sırasına göre sıralamış test kümesinin toplam uygunluk değeri bulunur.

Turnuva seçiminde uygunluk değerlerine göre belirlenen ebeveynler bu aşamadan sonra çaprazlama (crossover) ve mutasyon işlemlerine tabi tutularak ebeveynlerden farklı 2 çocuk elde edilmektedir. Çalışmamızda çarprazlama olasılığı 0.7, mutasyon olasılığı ise 0.15 olarak belirlenmiştir. Sıfır ile bir arasında rassal bir sayı üretildiğinde üretilen sayı çaprazlama aşamasında 0.7'den küçükse çaprazlama işlemi, mutasyon aşamasında rassal olarak üretilen sayı ise 0.15'ten küçük ise mutasyon işlemi gerçekleştirilmektedir.

Çalışmamızda sıra tabanlı çaprazlama yaklaşımı çaprazlama aşamasında kullanılmıştır.

Tablo 2.4'te sıra tabanlı çaprazlamayla ilgili küçük bir örnek verilmiştir. Bu örnekte test sayısı olarak ilk 8 test gösterim amaçlı alınmıştır.

Tablo 2.4 Çaprazlama ve Mutasyon

Sıra Tabanlı Çaprazlama	
Ebeveyn	**Çocuk**
T1,T2,T3,T4,T5,T6,T7,T8	T1,T7,T3,T4,T5,T2,T6,T9
T6,T9,T1,T7,T4,T3,T2,T5	T3,T5,T1,T7,T4,T6,T8,T2
Takas Mutasyonu	
Orijinal Kromozom	**Mutasyona Uğramış Kromozom**
T1,T2,T3,T4,T5,T6,T7,T8	T1,T2,T7,T4,T5,T6,T3,T8

Sıra tabanlı çaprazlamada ilk ebeveynin ilgili kesme noktalarındaki genleri ilk çocuğa aynen aktarılır, diğer genler ise ikinci ebeveynden ilk çocuğa aktarılır. İkinci ebeveynden ilk çocuğa kopyalanan genler ilk çocukta bulunuyorsa ikinci ebeveynde yer alan sıradaki gen ikinci çocuğa aktarılır ve ilk çocuktaki 100 genlik kromozom tamamlanana kadar bu işlem gerçekleştirilir. Aynı işlemler ikinci çocuk içinde gerçeklenir ve iki farklı çocuk sistem tarafından belirlenmiş olur.

Çaprazlama işleminden sonra bir diğer uygulanan operatör mutasyon işlemidir. Çalışmamızda mutasyon tipi olarak takas çaprazlama (Swap Mutation) yöntemi kullanılmıştır. Permütasyon gösteriminde takas çaprazlama iki genin kendi arasında değişimin göstermektedir.

Tablo 2.4'te 3'üncü ve 7'inci testlerin sırası kendi aralarında değişmiş ve böylece takas mutasyonu gerçeklenmiştir.

Çalışmamızda buraya kadar gördüğümüz adımları klasik genetik algoritma adımları olmakla birlikte genetik algoritmalarda yerel arama teknikleri kullanarak daha iyi sonuçlar alınabilir. Yerel arama teknikleri ilk popülasyon belirlemede, çaprazlamadan sonra, mutasyon işleminden sonra gibi farklı adımlardan sonra uygulanabilir. Çalışmamızda genetik algoritmayı iyileştirmek için mutasyon aşamasından sonra elde edilen 2 çocuk bireyin 20 farklı komşusuna bakılarak bireylerden daha yüksek uygunluk değeri olan komşu çocukların yerine popülasyona

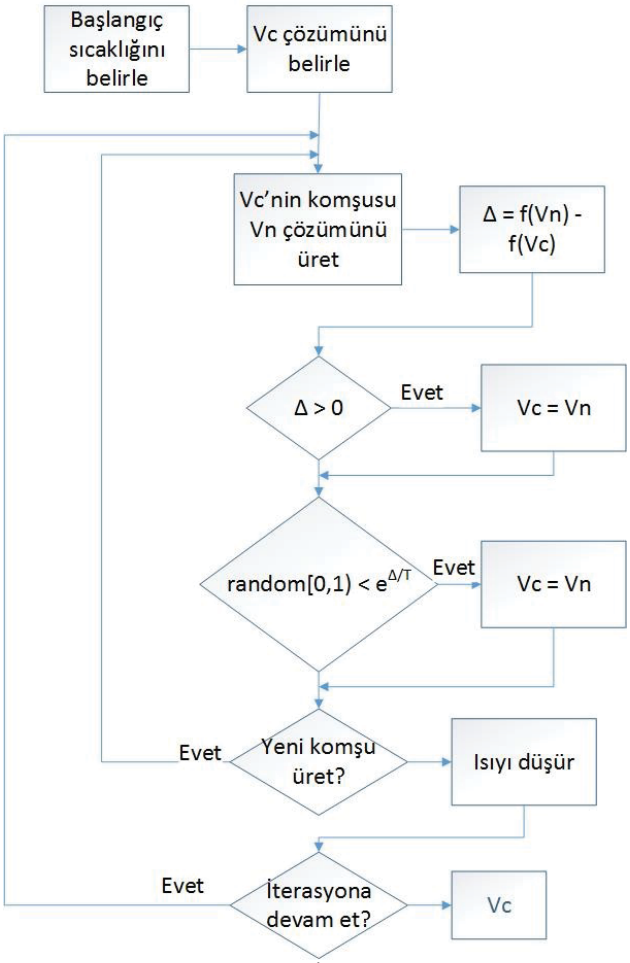
eklenmektedir. Komşuluk fonksiyonu olarak çalışmamızda takas mutasyonun aynısı uygulanmıştır. Takas mutasyonu sadece iki gen arasında değişim sağladığı için aynı zamanda orijinal kromozomun komşusu olarak da değerlendirilebilir. Eğer bir çocuğun komşularının uygunluk değeri çocuğun uygunluk değerinden küçükse popülasyona yine çocuk eklenmektedir.

Bölümün başında sistemimizin kararlı durum olduğunu belirtmiştik. En iyi komşusu bulduğumuz iki çocuk popülasyonda yer alan en kötü uygunluk değerine sahip iki bireyin yerine konmakta ve böylece hayatta kalan seçimi yapılmaktadır.

Isıl işlem algoritması, genetik algoritma ve diğer olasılıksal yaklaşımlar gibi en kısa zamanda doğru çözümü bulmayı amaçlar. Özellikle matematiksel formüllerle çözülemeyen kombinasyonel problemlerin en iyi çözümlerini bulmak için tercih edilmektedir [11,12,13]

Isıl işlem algoritması ile maksimizasyon problemlerini çözmek için kullanılan yöntemde sistemin yüksek bir sıcaklıktan yavaşça soğuduğu kabul edilmektedir. Soğudukça sistem maksimum noktaya doğru yaklaşmaktadır. Yaklaşılan noktanın yerel maksimum noktası olma ihtimali yüksek olduğu için sıcaklık değeri daha belirlenen eşik değerinin altına düşmemiş ise erişilen nokta rassal olarak başka bir nokta seçilmektedir. Bu rassallık sıcaklık düştükçe daha az uygulanarak maksimum noktasına erişilebilinmektedir.

Yaptığımız çalışmada ısıl işlem algoritmasını komşuluk kontrolü ile birlikte uygulayarak daha az iterasyon ile daha efektif sonuçlar elde edilmiştir. Sistem her iterasyonda bulduğu noktanın N kadar komşusuna da bakarak eğer kendisinden daha iyi bir komşusu var ise en iyi notkayı bulunan komşu ile değiştirmektedir.



Şekil 2.3 Isıl İşlem Algoritması

Isıl işlem algoritması Şekil 2.3'te verilmiştir. İlk adımda başlangıç çözümü ve başlangıç sıcaklığı belirlenir. İterasyon sayısı, kaç komşuya bakılacağı ve sıcaklık değerini düşürecek çarpan katsayı da belirlenir. Daha sonra iterasyon sayısı kadar işlem yapılır. Her işlemde bulunan ilk en iyi komşu ile eldeki en iyi çözüm güncellenir. Eğer hiçbir komşu daha iyi değilse yerel maksimum noktasına gelinme ihtimali olduğu için sistem bu noktadan çıkarılmalıdır. Bunun için bir rassal değer oluşturulur. Aynı zamanda her iterasyonda biraz daha düşürülen sıcaklığa göre belirlenen delta değeri oluşturulur. Eğer rassal değerden daha küçük bir delta oluşmuş ise sistem en iyi çözümünü değiştirir. Her iterasyon adımında sistem sıcaklığı soğuma katsayısı ile çarpılarak soğutulur. İterasyona devam etmek için şart, sıcaklığın minimum sıcaklık altına düşmemiş olması ve iterasyon sayısının aşılmamış olması gerekmektedir.

Çalışmamızda durma kriteri her iki algoritma için de 105 iterasyon (jenerasyon) olarak belirlenmiştir, değerlendirmeler bu sonuçlara göre yapılmıştır.

3. Deney ve Sonuçlar

Yaptığımız çalışmada 100 test ve bu testlerin tespit edebildiği 50, 100 ve 200 farklı hata durumları bulunmaktadır. Hangi hatanın hangi test tarafından tespit edildiği rassal olarak belirlenmiştir.

Çalışmamızda 50, 100 ve 200 rassal olarak belirlenmiş hata ayrı ayrı sistem tarafından test edilmiştir. Buna göre sistemimiz her bir durum için farklı ilk popülasyon değerleriyle 30 kere çalıştırılarak ortalama sonuçlar üzerinden değerlendirme yapılmıştır. Çalışmamızda sistemin başarısını karşılaştırabilmek adına 3 farklı yaklaşım kullanılmıştır. İlk yaklaşım ortaya çıkan hataların tamamı rassal olarak sıralanmış bir test kümesi tarafından kaçıncı test uygulandıktan sonra belirlendiğine dair olan yaklaşımdır (Yöntem 1, TTT). Yöntem 1'de önceliklendirmeler rassal olarak yapılmaktadır. Yöntem 2'de yerel arama ile iyileştirilmiş genetik algoritmayla (İGA) hataları belirleyen testler önceliklendirilmiştir. Yöntem 3'te ise ısıl işlem algoritması (IİA) ile tespit edilen en iyi test kümesinin önceliklendirme sonuçları elde edilmiştir.

Tablo 3.1'de uyguladığımız senaryonun sonuçları verilmektedir. Senaryoya göre 50, 100 ve 200 hata durumları oluşmuştur. Uygulanan ilk 20, 40, 60 ve 80 test ile ne kadar hatanın tespit edildiği gösterilmiştir. "Tümü" sütununda ise ilk kaç test ile tüm hataların tespit edildiği gösterilmiştir. Elde edilen veriler programın 30 kere çalıştırılması sonucunda elde edildiği için sonuçların ortalama değerlerini göstermektedir.

Tablo 3.1'deki sonuçlar incelendiğinde ratsgele test kümesinin oluşturulmasıyla 50 hata, 100 test içerisinde ortalama 94 test ile belirlenirken, iyileştirilmiş genetik algoritma ile bu ortalama değer 50,4'e, ısıl işlem algoritması ile 60,5 e düşmektedir. Elde edilen kazanımlar değerlendirildiğinde, 50 hata için, İGA ile ilk 40 test uygulandığında 45,3 hata tespiti yapılmaktadır. Aynı durumda IİA ile ilk 40 test uygulandığında 44,8 hata tespiti yapılmaktadır. Fakat rassal olarak belirlenen kümede ise bu değerin ortalaması 33 testte kalmaktadır. Tümü satırındaki sonuçları da doğrulayıcı veriler bu şekilde elde edilmiştir.

Tablo 3.1 Test Sayısı ve Belirlenen Hatalar

	Test Sayısı				
	20	**40**	**60**	**80**	**Tümü**
	50 Hata Durumu				
TTT	20,9	33	41	46,1	94
İGA	**33,83**	**45,3**	**49,9**	**50**	**50,4**
İİA	33	44,8	49,5	49,9	60,5
	100 Hata Durumu				
TTT	40,7	63,8	80,7	91,1	96,1
İGA	**60,6**	**84,4**	**95,5**	**99,9**	**69,6**
İİA	59,5	82,1	94,4	99,2	80,2
	200 Hata Durumu				
TTT	79,2	123,7	155,3	179,1	98,1
İGA	**112,9**	**161,2**	**186,4**	**197,9**	**82,8**
İİA	107,3	158,2	184,9	196,8	87,7

100 hata tanımlandığı durumda elde edilen ortalama verilere göre, rassal sıralama ile oluşturulan dizilimlerde 100 testten 96,1 test uygulandığında tüm hatalar tespit edilebilmektedir. İGA ile önceliklendirilme yapıldığında bütün hatalar ilk 69,6 test, IİA ile önceliklendirilme yapıldığında ise hataların tamamı ilk 80,2 test uygulandığında tespit edilebilmektedir. İlk 60 test uygulandığında İGA'nın 95,5, IİA ise 94,4 hatayı tespit edebildiği görülmektedir. Her ne kadar yakın olsalar da tüm hataların tespit edilmesi için önerilen çözümlerde ısıl işlem algoritmasının çözüm 80,2 test, genetik algoritmanın önerdiği çözüm 69,6 test uygulandığında başarılı olmaktadır. Buradan da genetik algoritmanın en başarılı sonucu üretebildiği gözlemlenmiştir.

Benzer şekilde 200 hata tanımlandığında rassal dizilim bütün hataları ortalama olarak 98,1 test ile tespit etmektedir. Genetik algoritma ile elde edilen önermede 82,8 test, ısıl işlem algoritması ile 87,7 test uygulandığında bütün hataların kontrolü sağlanabilmektedir. Algoritmaların bulduğu önermelerde, ilk 40 testin sonuçları ele alındığında, İGA ile 161,2 hata tespit edilirken, IİA ile 158,2 hatanın tespiti yapılabilmektedir.

Tablo 3.1'deki sonuçlar değerlendirildiğinde iyileştirilmiş genetik algoritma ile elde edilen değerlerde, ısıl işlem algoritmasına yakın sonuçlar elde edilmiş olsa da sürekli olarak daha iyi değerler üretebildiği gözlemlenmiştir.

Tablo 3.2 Uygulanan Testlere Göre Kazanım Yüzdeleri

	Uygulanan Test Sayısı			
	20	**40**	**60**	**80**
	50 Hata Durumu (Değerlik 278,97)			
TTT	43,50	67,81	83,34	92,95
İGA	**73,46**	**95**	**99,97**	**100**
İİA	73,01	93,77	99,71	99,95
	100 Hata Durumu (Değerlik 543,9)			
TTT	42,13	65,05	81,78	91,76
İGA	**63,99**	**88,22**	**98,11**	**99,99**
İİA	62,93	85,73	96,75	99,64
	200 Hata Durumu (Değerlik 1108,3)			
TTT	40,07	62,28	78,10	89,95
İGA	**59,11**	**82,91**	**94,59**	**99,56**
İİA	55,80	80,91	93,72	98,68

Tablo 3.2'de ise tespit edilen hata sayıları Tablo 3.1'de belirtilmiş olan TTT, İGA ve IİA uygulanmalarından sonra elde edilen en iyi genlerin toplam hata değerliliklerinden yüzde kaçını kapsadığı bilgisi yer almaktadır. Tablodaki verilere göre 50 hata bulunan senaryoda ilk 20 test uygulandığında TTT ile tüm hata değerliklerinin %43,50'si tespit edilirken, aynı durumda İGA ile %73,46'sı, IİA ile %73,01'i tespit edilebilmektedir.

Tablo 3.1 ve Tablo 3.2'de bulunan verilerin tamamı incelendiğinde rassal sıralamaya göre IİA ve İGA yöntemlerinden ikisinin de çok başarılı olduğu anlaşılırken bu iki faydalı yöntem arasından İGA'nın sürekli olarak daha yüksek kazanım sağladığı gözlemlenmiştir. Bununla birlikte hem iyileştirilmiş genetik algoritma hem de ısıl işlem algoritması ile elde edilen sonuçlar rassal yöntemden elde edilen sonuçlara oranla oldukça iyidir. Hata sayısı arttıkça İGA ve IİA'nın da doğal olarak performansı düşmektedir. Örneğin İGA 50 hatayı ortalama ilk 50,4 test ile tespit ederken, hata sayısı 4 katına çıktığında yani 200 olduğunda bu değer 82,8'e çıkmıştır; IİA ile 50 hata 60,5 test ile tespit edilebilirken, 200 hata 87,7 test ile tespit edilebilmektedir. Bu durum aynı zamanda rassal yöntem için de geçerlidir.

Bir sonraki çalışmada uygunluk fonksiyonunu iyileştirerek daha az sayıda test ile hata tespiti yapılmaya çalışılacaktır. Çalışmamızda uygunluk fonksiyonu en çok hatayı bulan testin ilk sıralarda uygulanma prensibine dayanmakla birlikte bulunan hatalar çıkarıldıktan sonra testlerin aynı prensibe göre sıralanmasıyla performansın artacağı düşünülmektedir.

Kaynaklar

1. Glass, Robert L.: Frequently forgotten fundamental facts about software engineering. Software, 18.3, pp. 112-111 (2001)
2. Rothermell, G., Harrold, M.J.: A safe, efficient regression test selection technique. ACM Transactions on Software Engineering and Methodology (TOSEM), 6.2, pp. 173-210 (1997)
3. Graves, Todd L., et al.: An empirical study of regression test selection techniques. Proceedings of the 20th international conference on Software engineering. IEEE Computer Society (1998)
4. Binkley, D.: Semantics guided regression test cost reduction. IEEE Transactions on Software Engineering, 23(8), pp. 498-516 (1997)
5. Shin, Y., Harman, M.: Pareto efficient multi-objective test case selection. Proceedings of the 2007 international symposium on Software testing and analysis, ACM (2007)
6. Sriraman, T., Gupta, N.: A concept analysis inspired greedy algorithm for test suite minimization. ACM SIGSOFT Software Engineering Notes. Vol. 31. No. 1. ACM (2005)
7. Eric, W.W.: A study of effective regression testing in practice. Proceedings The Eighth International Symposium On Software Reliability Engineering, IEEE (1997)
8. Eric, W.W.: Effect of test set minimization on fault detection effectiveness. Software Engineering, 1995, ICSE 1995, 17th International Conference on, IEEE (1995)
9. Rothermel, G., Untch, R.H., Chu, C., Harrold, M.J.: Prioritizing test cases for regression testing. Software Engineering, IEEE Transactions on, 27(10), pp. 929-948 (2001)
10. Li, Z., Harman, M., Hierons, R.M.: Search algorithms for regression test case prioritization. Software Engineering, IEEE Transactions on, 33(4), pp. 225-237 (2007)
11. Liu, Y., Xiong, S., Liu, H.: Hybrid simulated annealing algorithm based on adaptive cooling schedule for TSP. GEC '09 Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation, pp. 895-898 (2009)
12. Mansour, N., Bahsoon, R., Baradhi, G.: Empirical Comparison of Regression Test Selection Algorithms. Journal of Systems and Software, vol. 57, n. 1, pp. 79-90, Elsevier (2001)
13. Cingiz, M. Ozgur, Sefik Temel, and Oya Kalipsiz. "Test case prioritization with improved genetic algorithm." Signal Processing and Communications Applications Conference (SIU), 2014 22nd. IEEE, 2014.