# Öğrenme-Öğretme Temelli ve Armoni Arama Algoritmaları ile Test Verisi Üretimi

Bahriye Akay, Omur Sahin*

Erciyes Universitesi, Bilgisayar Muhendisligi, 38090
bahriye@erciyes.edu.tr
omur@erciyes.edu.tr

**Abstract.** The increase in the size of software made hard to manage the projects and caused software crisis such as late delivering, delivering with missing functions, not meeting the requirements. The main reason of software crisis is that testing is not carried out at each stage of software development cycle. When the test is left to post-delivery stages, both time and money budget increases exponentially. To conduct an efficient testing, all possibilities and senarios should be considered and test data supplied should produce maximum-coverage on the control-flow graph of the associated program. Meta-heuristic algorithms instead of applying an exhaustive search can produce high-quality test data that provide a maximum coverage. In this study, two of recent meta-heuristics, teaching-learning based optimization algorithm and harmony search algorithm has been applied to test data generations and their search ability has been analyzed on seven code fragments.

**Keywords:** Test data generation, TLBO Algorithm, Harmony Search Algorithm

**Özet.** Bilgisayar yazılımlarının boyutlarının büyümesi ile yönetilmeleri zorlaşmış, zamanında teslim edilememe, eksik fonksiyonla müşteriye sunma, gereksinimlerin karşılanmaması gibi yazılım krizleri ortaya çıkmıştır. Yazılım krizlerinin en büyük sebebi yazılımlarla ilgili testlerin en başından itibaren yapılmamasıdır. Her aşamada yapılması gereken testler teslimat sonrasına bırakıldığında projelerin zaman ve bütçe maliyetleri üstel olarak artmaktadır. Test süreçlerinin verimli bir şekilde yapılması için belli kriterler altında bütün durumların ve senaryoların test edilmesi gerekir. Meta-sezgisel algoritmalarda test verisi üretimi tüm olasılıkları denemeden akış grafiğinde maksimum kapsama sağlayacak verilen üretilemesini sağlar. Bu çalışmada test verisi üretimi için meta-sezgisel algoritmalardan öğrenme-öğretme temelli algoritma ile armoni algoritması yedi farklı kod parçacığı üzerinde çalıştırılmış ve arama yetenekleri incelenmiştir.

**Anahtar Kelimeler:** Test Verisi Üretimi, TLBO Algoritması, Armoni Arama Algoritması

# 1 Giriş

Diğer mühendislik ürünlerinde olduğu gibi, yazılımların doğru metodoloji üzerinden geliştirilmesiyle ortaya çıkan ürün daha kaliteli ve ekonomik olmaktadır. Standish Group'un yaptığı istatistikler incelendiğinde (Şekil 1), 2006 yılındaki projelerin %35'inin istenen zamanda, anlaşılan bütçe ve talep edilen gereksinimlerini karşılayacak şekilde tamamlandığı görülmektedir. %19'u bitmemiş yada iptal edilmiştir; %46 oranındaki projede ise geç teslimat, aşılmış bütçe veya sağlanmayan gereksinimler gibi krizler oluşmuştur.


**Fig. 1.** Projelerin başarı oranı. [31]

İçerisinde insan unsuru olmasından kaynaklı, uygulamalarda çeşitli hataların olması mümkündür. Uygulamaların teslimat ve kurulum öncesinde doğru çalıştığından ve kalitesinden emin olunmalıdır. Hataların erken safhalarda tespiti ile giderilme maliyeti oldukça düşük olacaktır; geç safhalarda tespit edildiğinde ise düzeltme maliyeti ve projenin toplam maliyeti de artacaktır (Şekil 2)[33]. Bu nedenle test işlemlerinin doğru bir şekilde erken safhalardan itibaren yapılması gerekmektedir.

Test işlemini için pek çok araç geliştirilmiştir. Bu araçlar çeşitli yöntemler ile test verileri üretmekte ve üretilen bu verileri çeşitli senaryolar ile denemektedir. Daha sonra elde ettiği sonuçlarla uygulamanın canlı sistemde davranışı hakkında fikir yürütebilmektedir. Üretilen bu test verileri yetersiz ise uygulamadaki hatalar görülemeyebilir ve uygulamanın yanlış veya verimsiz çalışmasına sebep olabilir. Bu yüzden test verileri yeterli ve pek çok durumu kontrol edebilecek şekilde üretilmelidir.

Çok sayıda test aracı olmasına rağmen bu yöntemlerin pek çoğunun sorunu bulunmaktadır. Kaynak kod içerisindeki sınıflar, döngüler, işaretçiler gibi karmaşık yapıların kullanımı bu sorunların bazılarıdır. Bu nedenle daha yüksek başarım (coverage'a dayalı) ve hız elde etmek için yapılan çalışmalar devam etmektedir.

Bu amaca yönelik olarak Webb Miller ve David Spooner arama tabanlı test verisi üretme yöntemini önermişlerdir [28]. Bu çalışmada kayan noktalı sayı tipindeki verilerin (float) giriş olarak verildiği programlar için sembolik çalıştırma ve

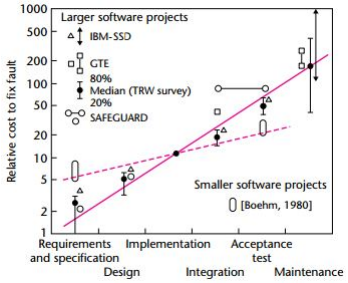


**Fig. 2.** Hataların bulunma fazlarına göre düzeltilme maliyetleri. [33]

sınırlamalı problem çözümünü içeren bir bir yöntem kullanışmıştır [23]. 90'lı yıllardan sonra Korel'in çalışmaları [14, 15] ve 1992'de Xanthakis'in genetik algoritma ile bu konuda bir çözüme ulaşmış olması [43] ile arama tabanlı test verisi üretimi popülerleşmiştir.

Arama tabanlı test verisi üretiminde bir amaç fonksiyonunun yönlendirmesi ile test verileri üretilir. İçyapısı bilinen programın test edildiği saydam kutu test tekniklerinden biri olan yapısal test [10, 24, 27, 38] için çeşitli çalışmalar yapılmıştır. Programın iç yapısının bilinmediği, girişin verilerek istenen çıkışın üretilmesini kontrol eden kara kutu test tekniklerinden biri olan fonksiyonel test [40] işlemi de oldukça popüler çalışma alanlarından biridir. Program üzerindeki olası yükün etkisini inceleyen stres testi, güvenlik testleri ve çeşitli performans testleri gibi fonksiyonel olmayan test [41] grubunda da çalışmalar bulunmaktadır. Sonlu durum makinalarındaki geçişlerle ifade edilen durum tabanlı test verisi üretimi de literatürde çalışılmıştır [6].

Literatürdeki mevcut çalışmalar incelendiğinde; test verilerinin üretiminde çeşitli optimizasyon algoritmalarının kullanılabildiği, elde edilen sonuçların başarımı yüksek son zamanlarda önerilmiş algoritmalarla daha da geliştirilebilir olduğu görülmüştür. Yapılan çalışmalardaki sonuçlar incelendiğinde bazı algoritmaların coverage türünden başarımının düşük olduğu, bazılarının ise hız açısından yetersiz olduğu görülmüştür. Bu çalışmada test verisi üretiminde literatürde henüz kullanılmamış güncel zeki optimizasyon algoritmalarından olan öğrenme-öğretme temelli algoritma ile armoni algoritması incelenmiştir.

İkinci bölümde arama tabanlı test verisi üretimi anlatılacak, üçüncü bölümde kullanılan meta-sezgisel algoritmalar kısaca özetlenecek, dördüncü bölümde yapılan deneysel çalışmalar anlatılacak ve son bölümde değerlendirmesi yapılacaktır.

## 2 Arama Tabanlı Test Verisi Üretimi

Arama tabanlı test verisi üretiminde (SBST) temel amaç üretilen test verilerinin kapsama metriğini maksimum yapmasıdır. Arama tabanlı yöntemlerden biri olan rastgele test verisi üretimi yöntemi [3] düşük maliyetli, kolay kısıtlara sahip kod parçaları için uygun test verileri üreten bir yöntemdir. Ancak zor

kısıtlı problemlerde oldukça kötü performans göstermektedir. Arama tabanlı test verisi üretimi ayrık bir problem olarak sınıflandırılabilir ve ayrık problemlerin çözümünde kullanılabilen meta-sezgisel algoritmalar test verisi üretiminde de kullanılabilmektedir. Harman ve Jones test verisi üretiminde kapsama metriğine bağlı olarak amaç fonksiyonu ile yönlendirilen meta-sezgisel algoritmaların kullanımının verimli sonuçlar üretebileceğini öne sürmüştür [8]. Meta-sezgisel algoritmalar problemin karakteristiğinden bağımsız olarak amaç fonksiyonu elde edilebilen bütün problemlerde kullanılabilmektedir. Arama tabanlı test verisi üretimi hakkında yapılan çalışmalar Harman ve Jones [8], McMinn [25], Afzal ve ark. , Räihä [30], McMinn [23], Harman ve ark. [9], Harman ve ark. [11] tarafından derlenmiştir. Bütün bu derlemelerde arama tabanlı yazılım mühendisliği alanında yapılabilecek çalışmaların mevcut olduğu ve her geçen gün popüler hale geldiği görülmektedir.

Literatürde doğal yaşamdan esinlenerek önerilen çok sayıda meta-sezgisel bulunmaktadır. Parçacık sürü optimizasyonu (PSO) [13], diferansiyel gelişim (DE) [35], yapay arı koloni (ABC) [12] ve ateş böceği (FA) algoritmaları [44] en popüler meta-sezgisel algoritmalar arasında gösterilmektedir.

Meta-sezgisel algoritmalar amaç fonksiyonları doğrultusunda arama yaparlar. Bu nedenle iyi tasarlanmış bir amaç fonksiyonu algoritmanın optimum değeri hızlı ve doğru bir şekilde bulmasına yardımcı olur [8]. Çeşitli çalışmalarda kullanılan amaç fonksiyonları; code/statement coverage [37], yol tabanlı kriter [18, 20, 19, 5, 22, 21, 36, 34], kenar (edge) kapsamı [16], veri akış kapsamı (data flow coverage) [32], dal kapsamı (branch coverage) [42, 2], dal uzaklığı (branch distance) [26, 4], approximation level + branch distance [17] şeklinde verilebilir. Approximation level + branch distance metriklerini temel alan amaç fonksiyonu en sık kullanılan amaç fonksiyonlarından biridir.

Approximation level + branch distance (Eşitlik 1) amaç fonksiyonu branch distance (Eşitlik 2) ve approximation level metriklerinin birleşiminden oluşmaktadır.

$$fitness_{ALBD} = approximation\ level + normalize(brach\ distance) \quad (1)$$

$$normalize(branch\ distance) = 1 - 1.001^{-branch\ distance} \quad (2)$$

Approximation level çalıştırılması gereken yol ile çalıştırılan yolun kıyaslayarak çalıştırılamayan yolların toplamını veren bir metriktir [1]. Branch distance metriği ise Tracey'nin ortaya attığı [39] ve Tablo 1'da verilen yöntem ile hesaplanmaktadır. Branch distance ilgili dala girmek için ne kadar uzakta olunduğunu belirtir. Tablo 1'da görülen $K$ değeri sonucun her zaman pozitif olmasını sağlayacak sabit bir değerdir.

## 3 Meta-sezgisel Algoritmalar

Meta-sezgisel algoritmalar, herhangi bir amacı sezgisel yöntemlerle en iyileme işlemini gerçekleştiren algoritmalardır. En iyileme yaparken kullandığı yöntem

**Table 1.** Tracey'nin önermiş olduğu branch distance fonksiyonu. Kod parçacığında ilk sütunda görülen değer ile karşılaşıldığında 2. sütundaki değer hesaplanmaktadır. [39]

| | Amaç Fonksiyonu |
|---|---|
| Boolean | if $TRUE$ then 0 else $K$ |
| $a = b$ | if $abs(a - b) = 0$ then 0 else $abs(a - b) + K$ |
| $a \neq b$ | if $abs(a - b) \neq 0$ then 0 else $abs(a - b) + K$ |
| $a < b$ | if $a < b$ then 0 else $a - b + K$ |
| $a \leq b$ | if $a \leq b$ then 0 else $a - b + K$ |
| $a > b$ | if $a > b$ then 0 else $b - a + K$ |
| $a \geq b$ | if $a \geq b$ then 0 else $b - a + K$ |

optimal sonucu garanti etmemekle birlikte optimuma yakın sonuç üretmeye yöneliktir. Bu çalışmada henüz literatürde bu alana uygulanmamış öğretme-öğrenme temelli ve armoni arama algoritmaları kullanılmıştır.

### 3.1 Öğretme-Öğrenme Temelli Optimizasyon Algoritması

Öğrenme-Öğretme temelli Optimizasyon Algoritması (Teaching-Learning Based Optimization Algorithm, TLBO) [29] bir sınıftaki öğrencilerin öğrenmesinde öğreticinin etkisini temel alarak çalışan bir algoritmadır. Öğrencilerin başarımının ölçülmesi notlar üzerinden gerçekleştirilir. Öğretmen ise o sahadaki en yetkin kişi olarak nitelendirilebilir. İyi bir öğretmenin öğrencileri doğru yönlendirerek yüksek notlar alabileceği fikri üzerine geliştirilmiştir. Ayrıca sınıf içi etkileşimle öğrenciler arasındaki bilgi aktarımı da dikkate alınmıştır. Burada öğrenciler optimizasyon probleminin olası çözümlerine, öğrencilerin notları da uygunluk fonksiyonuna karşılık gelmektedir. Popülasyondaki en iyi birey öğretmen olarak seçilir. Algoritma öğretici fazı ve öğrenci fazı olmak üzere iki aşamadan oluşmaktadır.

**Öğretici Fazı** Bir öğretmen öğrencilerine tüm bildiklerini aktararak onları kendi seviyesine getirmesi başka dış hususlar da devreye girdiğinden mümkün olamamakta. Dolayısıyla bu süreçte kontrol edilemeyen rastgelelik arz eden etmenler bulunmaktadır. $M_i$, sınıfın yani popülasyonun ortalama değeri, $T$ öğretici (yani en iyi çözümü), $T_f$ öğretme faktörü ve $r$ [0, 1] aralığında rastgele bir reel sayı olmak üzere $X_i$ çözümünün yeni değeri ($X'_i$), Eşitlik 3 ile belirlenir:

$$X'_i = X_i + r_i(T - T_f * M_i) \tag{3}$$

$T_f$ öğretme faktörü Eşitlik 4 ile 1 yada 2 olarak belirlenen bir parametredir.

$$T_f = round[1 + rand(0, 1)(2 - 1)] \tag{4}$$

Bu ifade öğreticinin öğrencinin gelişimi üzerindeki etkisini yansıtmaktadır. Yeni çözüm eskisinden daha iyi ise diğeri yerine popülasyona dahil edilir, öğrenci bilgisini güncellemiş olur.

**Öğrenci Fazı** Öğrencilerin gelişiminin öğretmene bağlı olmasının yanı sıra diğer öğrencilerle olan etkileşimine de bağlıdır. Öğrenci fazında rastgele seçilen bir sınıf arkadaşı o anki öğrenciden daha iyi ise öğrenci bu bireyden Eşitlik 5 ile faydalanır:

$$X_i' = X_i + r_i(X_i - X_j) \tag{5}$$

O anki öğrenci rastgele seçilen arkadaştan daha iyi ise Eşitlik 6 kullanılır:

$$X_i' = X_i + r_i(X_j - X_i) \tag{6}$$

Bu faz sonrasında yeni öğrenci bilgisi ($X_i'$) eski değerinden ($X_i$) daha iyi bir uygunluk değerine sahipse öğrenci güncellenir.

Bu iki faz durdurma kriteri sağlanıncaya kadar tekrarlanır.

### 3.2 Armoni Arama Algoritması

Armoni Arama (Harmony Search, HS) algoritması [7] müzisyenlerin beste yapma süreçlerini modelleyen bir meta-sezgisel algoritmadır. Çözümler müzisyenlere, notaların armonisi de çözümün uygunluk değerine karşılık gelir. Algoritmanın temel adımları şu şekildedir:

- Adım 1: Problem ve algoritma parametrelerinin ilklendirilmesi
- Adım 2: Armoni repertuvarının ilklendirilmesi
- Adım 3: Yeni bir armoni bestelenmesi
- Adım 4: Armoni repertuvarının güncellenmesi
- Adım 5: Durma kriteri sağlanıncaya kadar Adım 2-5'in tekrarlanması

HS algoritmasında, her iterasyonda yeni bir beste üretilir ve bu beste armoni repertuvarındaki en kötü armoni ile değiştirilir. Besteleme sürecinde yeni armoni Eşitlik 7 ile üretilir:

$$\begin{array}{l} if(rand \geq p_{hcmr}) \\ x_j' = x_j^{\min} + (x_j^{\max} - x_j^{\min}) * rand \\ else \\ \{ \\ x_j' = x_{\text{int}(rand*SN)+1,j} \\ if(rand < p_{par}) \\ \{ \\ x_j' = x_j' \pm bw * rand \\ \} \\ \} \end{array} \tag{7}$$

Burada $SN$ repertuvar büyüklüğü, $p_{hcmr}$ repertuvardan seçme oranı, $p_{par}$ perde ayar oranı, $bw$ perde ayarlamasında olabilecek maksimum değişimi ifade eden bantgenişliği mesafesidir.

## 4 Deneysel Çalışmalar

Yapılan bu çalışmada triangle, quadratic equation, even-odd, largest number, remainder, leap year ve mark problemleri kullanılmıştır. Bu problemlerin karakteristikleri Tablo 2'de görülmektedir. Bütün problemler 20 popülasyon büyüklüğü ile her gerekli yol için 250 çevrim olarak koşulmuştur. Algoritma maksimum çevrim sayısına ulaştığında veya %100 kapsama miktarına eriştiğinde durmaktadır. Her bir problem için sonuçlar 2.3 GHz i7 işlemci ve 8 GB Ram özelliklerine sahip bilgisayarda 30 defa koşularak alınmıştır. Uygulama MATLAB ve PYTHON dilinde yazılmıştır. Elde edilen sonuçlar ilk satırda ortalama ile standart sapma ikinci satırda ise medyan olarak verilmiştir.

- **Triangle**: Bu program aldığı üç değerin üçgen oluşturup oluşturmadığına bakmaktadır. Eğer üçgen oluşturuyorsa da eşkenar, ikizkenar veya çeşitkenar üçgen olup olmadıklarına karar vermektedir.
- **Quadratic Equation**: Bu program aldığı üç değere göre $ax^2 + bx + c$ formatında olup olmadığına bakmakta, daha sonra diskriminant hesabını yaparak denklemin köklerini hesaplamaktadır.
- **Even-Odd**: Bu program girilen değerin tek mi çift mi olduğuna karar vermektedir.
- **Largest Number**: Bu program girilen üç değer arasında en büyük değeri bulmaktadır.
- **Remainder**: Bu program aldığı iki değere bakarak bölenin sıfır olup olmadığına bakarak kalan hesabı yapmaktadır.
- **Leap Year**: Bu program girilen değerin artık yıl olup olmadığına bakmaktadır.
- **Mark**: Bu program aldığı üç değere göre ortalama hesaplamaktadır. Daha sonra ise elde ettiği ortalamaya göre beş farklı kategoride (A, B, C, D, E) kümelemektedir.

**Table 2.** Problemlerin karakteristikleri

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

Kullanılan uygulama mimarisi program analizi, yol seçici ve test verisi üreticisi parçalarından oluşmaktadır (Şekil 3). Program analizi, kaynak kodu test verisi üretimi için anlamlı hale getirmektedir. Programın kaynak kodu, kontrol akış

grafiği (CFG) adı verilen ve programın akışının graf üzerinde gösterimini sağlayan bir yapıya çevrilmektedir. Oluşturulan bu CFG başlangıç ve bitiş düğümlerinin tekil olduğu yönlendirilmiş graftır. Oluşturulan bu CFG üzerinden algoritmalar için bir amaç fonksiyonu hesaplanmaktadır.

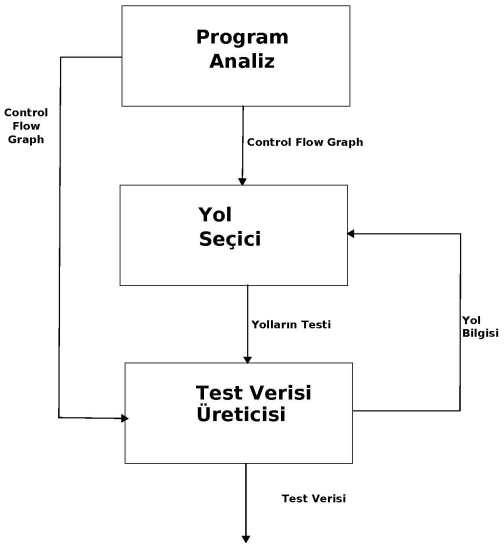


**Fig. 3.** Test verisi üretimi için kullanılan mimari

Yapılan koşmalar sonucu elde edilen sonuçlar Tablo 3'da verilmiştir. Bu sonuçlara göre en kolay problem olan even-odd probleminde her iki algoritma da aynı başarımda sonuçlar üretmiştir. Aynı şekilde bir başka basit problem olan largest number probleminde de her iki algoritma aynı başarıma ulaşmıştır. Aynı iterasyon sayısında çalışan bu iki algoritma incelendiğinde medyan değerlerine göre HS algoritmasının TLBO algoritmasından daha hızlı çalıştığı görülmektedir. Remainder probleminde ise her iki algoritma da %100 başarım elde etse de HS algoritması daha fazla iterasyona ihtiyaç duymuş bu da yavaş kalmasına neden olmuştur. Leap year, mark ve triangle problemler incelendiğinde her iki algoritma da medyan değerlerine göre maksimum başarıma erişmiştir. Ortalama değerler incelendiğinde ise TLBO algoritmasının HS algoritmasına göre üstünlüğü görülmektedir. Bu da bazı koşmalarda HS algoritmasının maksimum başarıma erişemediğini göstermektedir. Quadratic equation problemi incelendiğinde ise hem medyan hem de ortalama değerlere göre TLBO algoritmasının başarısı görülmektedir.

**Table 3.** TLBO ve HS algoritmaları karşılaştırma sonuçları

| | | Kapsama | İterasyon | Zaman |
|---|---|---|---|---|
| TLBO | Even Odd | 100.0000 ∓ 0.0000 (100.0000) | 2.0000 ∓ 0.0000 (2.0000) | 0.0372 ∓ 0.0291 (0.0318) |
| | Largest Number | 100.0000 ∓ 0.0000 (100.0000) | 4.0000 ∓ 0.0000 (4.0000) | 0.0748 ∓ 0.0280 (0.0692) |
| | Leap Year | 100.0000 ∓ 0.0000 (100.0000) | 21.0333 ∓ 27.0038 (11.0000) | 0.2186 ∓ 0.2454 (0.1231) |
| | Mark | 100.0000 ∓ 0.0000 (100.0000) | 512.4000 ∓ 79.3137 (505.0000) | 5.3441 ∓ 0.8162 (5.2652) |
| | Quadratic Equation | 92.0000 ∓ 4.8423 (90.0000) | 275.5000 ∓ 112.5141 (272.0000) | 2.7435 ∓ 1.0981 (2.7063) |
| | Remainder | 100.0000 ∓ 0.0000 (100.0000) | 2.0667 ∓ 0.2537 (2.0000) | 0.0400 ∓ 0.0288 (0.0340) |
| | Triangle | 100.0000 ∓ 0.0000 (100.0000) | 45.1667 ∓ 38.1658 (36.0000) | 0.5147 ∓ 0.3882 (0.4177) |
| HS | Even Odd | 100.0000 ∓ 0.0000 (100.0000) | 2.0000 ∓ 0.0000 (2.0000) | 0.0416 ∓ 0.1098 (0.0215) |
| | Largest Number | 100.0000 ∓ 0.0000 (100.0000) | 4.0000 ∓ 0.0000 (4.0000) | 0.0559 ∓ 0.0513 (0.0465) |
| | Leap Year | 93.3333 ∓ 8.9115 (100.0000) | 153.5667 ∓ 88.5405 (147.5000) | 0.6123 ∓ 0.3516 (0.5771) |
| | Mark | 99.4667 ∓ 2.0297 (100.0000) | 279.5333 ∓ 89.9646 (272.0000) | 1.3765 ∓ 0.4257 (1.3366) |
| | Quadratic Equation | 84.0000 ∓ 4.9827 (80.0000) | 444.0667 ∓ 80.9269 (500.0000) | 2.0352 ∓ 0.3679 (2.2647) |
| | Remainder | 100.0000 ∓ 0.0000 (100.0000) | 15.1333 ∓ 20.6994 (9.5000) | 0.0830 ∓ 0.0904 (0.0532) |
| | Triangle | 99.5960 ∓ 1.5376 (100.0000) | 235.2333 ∓ 99.8615 (200.5000) | 1.1494 ∓ 0.4952 (0.9725) |

## 5 Değerlendirme

Test verisi üretimi problemi NP-Hard türünde oldukça zor bir problem türüdür. Deterministik yöntemler ile çözülmesi oldukça zordur ve çok uzun sürelerde gerçekleşebilmektedir. Programdaki döngüler, işaretçiler, sınıf yapıları bu problemi çok daha zor hale getirmektedir. Sezgisel algoritmalar ile kabul edilebilir zaman dilimlerinde makul çözümler üretmek mümkündür. Bu nedenle arama tabanlı test verisi üretiminin sürekli popülerlik kazanmaktadır.

Even odd ve largest gibi kolay problemlerde TLBO ve HS algoritmaları iyi performans göstermişlerdir. Remainder probleminde de her iki algoritma medyan ve ortalama değerlere göre maksimuma erişmişlerdir fakat TLBO algoritması HS algoritmasından daha az çevrime ihtiyaç duymuştur. Leap year, mark, quadratic ve triangle problemlerinin hepsinde de TLBO algoritması daha iyi sonuç elde etmiştir. Bütün algoritmalar incelendiğinde TLBO algoritmasının HS algoritmasından daha iyi sonuç ürettiği görülmektedir.

Gelecek çalışmalarda problem sayısı ve kullanılan algoritma sayısı artırılabilir. Aynı zamanda bu algoritmaların paralelleştirilmiş performansları da incelenebilir.

## References


1. André Baresel, Harmen Sthamer, and Michael Schmidt. Fitness function design to improve evolutionary structural testing. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '02, pages 1329–1336, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.
2. R. Landa Becerra, R. Sagarna, and X. Yao. An evaluation of differential evolution in software test data generation. In *2009 IEEE Congress on Evolutionary Computation*. Institute of Electrical & Electronics Engineers (IEEE), may 2009.
3. D. L. Bird and C. U. Munoz. Automatic generation of random self-checking test cases. *IBM Syst. J.*, 22(3):229–245, September 1983.
4. S.S. Dahiya, J.K. Chhabra, and S. Kumar. Application of artificial bee colony algorithm to software testing. In *Software Engineering Conference (ASWEC), 2010 21st Australian*, pages 149–154, April 2010.
5. Surender Singh Dahiya, Jitender Kumar Chhabra, and Shakti Kumar. Pso based pseudo dynamic method for automated test case generation using interpreter. In *Proceedings of the Second International Conference on Advances in Swarm Intelligence - Volume Part I*, ICSI'11, pages 147–156, Berlin, Heidelberg, 2011. Springer-Verlag.
6. Karnig Derderian, Robert M. Hierons, Mark Harman, and Qiang Guo. Automated unique input output sequence generation for conformance testing of fsms. *Comput. J.*, 49(3):331–344, May 2006.
7. Zong Woo Geem, Joong Hoon Kim, and G.V. Loganathan. A new heuristic optimization algorithm: Harmony search. *SIMULATION*, 76(2):60–68, feb 2001.
8. Mark Harman and Bryan F Jones. Search-based software engineering. *Information and Software Technology*, 43(14):833 – 839, 2001.
9. Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.*, 45(1):11:1–11:61, December 2012.
10. Mark Harman and Phil McMinn. A theoretical and empirical study of search-based testing: Local, global, and hybrid search. *IEEE Trans. Softw. Eng.*, 36(2):226–247, March 2010.
11. Mark Harman, Phil McMinn, JerffesonTeixeira de Souza, and Shin Yoo. Search based software engineering: Techniques, taxonomy, tutorial. In Bertrand Meyer and Martin Nordio, editors, *Empirical Software Engineering and Verification*, volume 7007 of *Lecture Notes in Computer Science*, pages 1–59. Springer Berlin Heidelberg, 2012.
12. D. Karaboga. An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.
13. J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *1995 IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948", 1995.
14. B. Korel. Automated software test data generation. *IEEE Trans. Softw. Eng.*, 16(8):870–879, August 1990.
15. B Korel, H Wedde, and R Ferguson. Dynamic method of test data generation for distributed software. *Information and Software Technology*, 34(8):523 – 531, 1992.

16. Soma Sekhara Babu Lam, M L Hari Prasad Raju, Uday Kiran M, Swaraj Ch, and Praveen Ranjan Srivastav. Automated generation of independent paths and test suite optimization using artificial bee colony. *Procedia Engineering*, 30:191 – 200, 2012. International Conference on Communication Technology and System Design 2011.
17. Gentiana Ioana Latiu, Octavian Augustin Cret, and Lucia Vacariu. Automatic test data generation for software path testing using evolutionary algorithms. In *Proceedings of the 2012 Third International Conference on Emerging Intelligent Data and Web Technologies*, EIDWT '12, pages 1–8, Washington, DC, USA, 2012. IEEE Computer Society.
18. D. Jeya Mala and V. Mohan. Abc tester artificial bee colony optimization for software test suite optimization. *IJSE International Journal Of Software Engineering*, 2(2):15–48, 2009.
19. D. Jeya Mala, V. Mohan, and M. Kamalapriya. Automated software test optimisation framework – an artificial bee colony optimisation-based approach. *IET Software*, 4(5):334, 2010.
20. D.J. Mala, M. Kamalapriya, R. Shobana, and V. Mohan. A non-pheromone based intelligent swarm optimization technique in software test suite optimization. In *Intelligent Agent Multi-Agent Systems, 2009. IAMA 2009. International Conference on*, pages 1–5, July 2009.
21. Ruchika Malhotra, Chand Anand, Nikita Jain, and Apoorva Mittal. Article: Comparison of search based techniques for automated test data generation. *International Journal of Computer Applications*, 95(23):4–8, June 2014.
22. Ruchika Malhotra and Manju Khari. Test suite optimization using mutated artificial bee colony. In *Proc. of Int. Conf. on Advances in Communication Network and Computing, CNC*, pages 45 – 54, 2014.
23. P. McMinn. Search-based software testing: Past, present and future. In *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*, pages 153–163, March 2011.
24. P. McMinn, M. Harman, K. Lakhotia, Y. Hassoun, and J. Wegener. Input domain reduction through irrelevant variable removal and its effect on local, global, and hybrid search-based structural test data generation. *Software Engineering, IEEE Transactions on*, 38(2):453–477, March 2012.
25. Phil McMinn. Search-based software test data generation: A survey: Research articles. *Softw. Test. Verif. Reliab.*, 14(2):105–156, June 2004.
26. Xiao mei Zhu and Xian feng Yang. Software test data generation automatically based on improved adaptive particle swarm optimizer. In *2010 International Conference on Computational and Information Sciences*. Institute of Electrical & Electronics Engineers (IEEE), dec 2010.
27. C. C. Michael, G. McGraw, and M. A. Schatz. Generating software test data by evolution. *IEEE Trans. Softw. Eng.*, 27(12):1085–1110, December 2001.
28. W. Miller and D.L. Spooner. Automatic generation of floating-point test data. *Software Engineering, IEEE Transactions on*, SE-2(3):223–226, Sept 1976.
29. R.V. Rao, V.J. Savsani, and D.P. Vakharia. Teaching-learning-based optimization: An optimization method for continuous non-linear large scale problems. *Information Sciences*, 183(1):1 – 15, 2012.
30. Outi Räihä. A survey on search-based software design. *Computer Science Review*, 4(4):203 – 249, 2010.
31. D. Rubenstein. Standish group report: There's less development chaos, 2007.

32. H M Rai Sanjay Singla, Dharminder Kumar and Priti Singla. A hybrid pso approach to automate test data generation for data flow coverage with dominance concepts. *International Journal of Advanced Science and Technology*, 37, 2011.
33. Stephen Schach. *Object-Oriented and Classical Software Engineering*. McGraw-Hill Education, 2010.
34. Praveen Ranjan Srivatsava, B. Mallikarjun, and Xin-She Yang. Optimal test sequence generation using firefly algorithm. *Swarm and Evolutionary Computation*, 8:44 – 53, 2013.
35. Rainer Storn and Kenneth Price. Differential evolution - a simple and efficient heyristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.
36. Dr.Bharti Suri and Prabhneet Kaur. Path based test suite augmentation using artificial bee colony algorithm. *International Journal For Research in Applied Science and Engineering Technology*, 2:156–164, 2014.
37. Shailesh Tiwari, K.K. Mishra, and A.K. Misra. Test case generation for modified code using a variant of particle swarm optimization (PSO) algorithm. In *2013 10th International Conference on Information Technology: New Generations*. Institute of Electrical & Electronics Engineers (IEEE), apr 2013.
38. Paolo Tonella. Evolutionary testing of classes. *SIGSOFT Softw. Eng. Notes*, 29(4):119–128, July 2004.
39. N. Tracey, J. Clark, K. Mander, and J. McDermid. An automated framework for structural test-data generation. In *Proceedings of the 13th IEEE International Conference on Automated Software Engineering*, ASE '98, pages 285–, Washington, DC, USA, 1998. IEEE Computer Society.
40. Joachim Wegener and Oliver Bühler. Evaluation of different fitness functions for the evolutionary testing of an autonomous parking system. In Kalyanmoy Deb, editor, *Genetic and Evolutionary Computation – GECCO 2004*, volume 3103 of *Lecture Notes in Computer Science*, pages 1400–1412. Springer Berlin Heidelberg, 2004.
41. Joachim Wegener and Matthias Grochtmann. Verifying timing constraints of real-time systems by means of evolutionary testing. *Real-Time Systems*, 15(3):275–298, 1998.
42. Andreas Windisch, Stefan Wappler, and Joachim Wegener. Applying particle swarm optimization to software testing. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, pages 1121–1128, New York, NY, USA, 2007. ACM.
43. S. Xanthakis, C. Ellis, C. Skourlas, A. Le Gal, S. Katsikas, and K. Karapoulios. Application of Genetic Algorithms to Software Testing. In *International Conference on Software Engineering*, 1992.
44. Xin-She Yang. *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2008.