YALOVA ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

GENETİK ALGORİTMA TABANLI AKILLI TEST SAYFASI ÜRETİMİ

YÜKSEK LİSANS TEZİ

Ufuk TÜL

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

HAZİRAN 2018

YALOVA ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

GENETİK ALGORİTMA TABANLI AKILLI TEST SAYFASI ÜRETİMİ

YÜKSEK LİSANS TEZİ

Ufuk TÜL
135105001

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Dr. Öğr. Üyesi Adem TUNCER

HAZİRAN 2018

YALOVA Üniversitesi Fen Bilimleri Enstitüsü'nün 135105001 numaralı Yüksek Lisans Öğrencisi **Ufuk TÜL**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"GENETİK ALGORİTMA TABANLI AKILLI TEST SAYFASI ÜRETİMİ"** başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı : **Dr. Öğr. Üyesi Adem TUNCER**
Yalova Üniversitesi .....

Jüri Üyeleri : **Dr. Öğr. Üyesi Adem TUNCER**
Yalova Üniversitesi .....

Dr. Öğr. Üyesi Osman Hilmi KOÇAL
Yalova Üniversitesi .....

Dr. Öğr. Üyesi Suhap ŞAHİN
Kocaeli Üniversitesi .....

Teslim Tarihi : **15 Mayıs 2018**
Savunma Tarihi : **20 Haziran 2018**

Çok değerli aileme,

ÖNSÖZ

Teknolojik gelişmeler ve her çeşit verinin dijital ortamda saklanmasıyla hayat daha da kolaylaşmakta, insan iş gücü ve zaman kaybı azaltılmaktadır. Eğitim ve öğretim alanında da istenilen kriter ve özelliklere göre test sayfası oluşturma işleminin kolay, etkin ve hızlı bir şekilde yapılması önemli bir hale gelmektedir. Bu tez çalışmasında test sayfası oluşturma problemine çözüm için genetik algoritma kullanılmış ve istenen özelliklerde test sayfaları üretebilen web tabanlı bir uygulama geliştirilmiştir.

Tez çalışması boyunca bilgi ve tecrübelerini aktaran, zaman ve mekan farketmeden her zaman ulaşabildiğim, beni doğru şekilde motive eden, titizlikle çalışarak ilham veren ve bana yardımcı olan çok değerli tez danışmanım Dr. Öğr. Üyesi Adem TUNCER'e teşekkürlerimi bir borç bilirim.

Ayrıca, bu süreçte bana verdikleri destek için tüm aileme, göstermiş olduğu sabır, anlayış ve her türlü yardımlarından dolayı eşim Havva ve oğlum Kerem'e teşekkürlerimi ve sevgilerimi sunarım.

Mayıs 2018

Ufuk Tül
Bilgisayar Mühendisi

İÇİNDEKİLER

Sayfa

KISALTMALAR

e-YDS **:** Elektronik Yabancı Dil Sınavı
FIFO **:** First In First Out
GA **:** Genetik Algoritma
IEEE **:** The Institute of Electrical and Electronics Engineers
IELTS **:** International English Language Testing System
KKA : Karınca Koloni Algoritması
PSO : Parçacık Sürü Optimizasyonu
TOEFL **:** Test of English as a Foreign Language

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

GENETİK ALGORİTMA TABANLI AKILLI TEST SAYFASI ÜRETİMİ

ÖZET

Son yıllarda teknolojinin hızlı gelişimine bağlı olarak, verilerin hayatın her alanında olduğu gibi eğitim alanında da elektronik ortamda etkin bir şekilde saklanması ve kullanılması mümkün hale gelmiştir. Elektronik ortamların ve kâğıtsız çözümlerin giderek yaygınlaştığı bu çağda teknolojinin sağladığı yenilikler her geçen gün hayatımızı daha da kolaylaştırmaktadır. Eğitim sisteminin bir parçası olan ölçme-değerlendirme çalışmaları için uygulanan sınavlar da elektronik ortamlarda hazırlanarak zaman ve mekandan bağımsız olarak yapılabilmekte ve böylece geniş kitlelere kolay ulaşabilme kabiliyeti sunabilmektedir. Sınavlar için hazırlanan test soruları da her birinin kendine ait çeşitli özellikleriyle birlikte elektronik ortamda saklanabilmekte ve kullanılabilmektedir.

Eğitim ve öğretim alanında elektronik soru bankalarının kullanılmasıyla birlikte, istenilen özelliklerde test sayfalarının hazırlanabilmesi ihtiyacı ortaya çıkmaktadır. Akıllı test sayfalarının oluşturulması konusunda farklı yöntem ve algoritmalar kullanılarak insan yerine bilgisayar vb. çalışmakta ve böylece zamandan tasarruf sağlanırken istenilen seviyede kaliteli ve etkin test sayfaları oluşturulabilmektedir. Elektronik ortamda bulunan bir soru bankasından istenen kriterlerde test sayfası oluşturma işlemi için geleneksel algoritma ve yöntemlerinin kullanıldığı çalışmalar mevcuttur. Fakat bu yöntemlerin uygulanması genellikle çözüm süresini uzatır, test sayfasının geçmişte oluşturulan sayfalarla aynı olması gibi durumlar meydana getirebilir ve test sayfası üretimi için verimi düşürebilir. Matematiksel veya geleneksel yöntemlerle çözümü zor olan veya çözüm süresi uzun süren problemler için alternatif olarak sezgisel yöntemler sıkça kullanılmaktadır. Bu problemler yapay zeka alanında kullanılmakta olan sezgisel optimizasyon yöntemleri ile aşılabilmekte, kısıtlı sorulara sahip olan bir soru bankasında bile istenilen kriterlere göre optimum sonuçlar elde edilebilmektedir. Sezgisel optimizasyon yöntemleriyle her zaman en iyi çözümü üretme gibi bir amaç olmadan, rastgele seçilen sorular üzerinden gidilerek çözüm en iyiye yakınlaştırılabilmektedir. İstenilen çok sayıda kriter değerlendirilerek çözüm için optimum sonuç bulunabilmektedir.

Tez çalışmasında çoklu kısıtlara sahip olan test sayfası oluşturma problemine sezgisel bir yaklaşım ile çözüm aranmış, aynı zamanda insan iş gücü ve zaman kaybının azaltılması amaçlanmıştır. Test sayfası problemine uyarlanmış genetik algoritma istenilen kriterlerde ve özelliklerde test sayfası oluşturulması işlemi hızlı ve etkin bir şekilde sağlanmıştır. Çalışmada test sayfası oluşturma işleminin kullanıcılar tarafından da kolay bir şekilde gerçekleştirilebilmesi amacıyla web tabanlı bir uygulama yazılımı gerçekleştirilmiştir. Çalışmada kullanılan genetik algoritma ile standart genetik algoritma sonuçları karşılaştırmalı olarak verilmiş ve çalışmada kullanılan genetik algoritmanın daha iyi sonuçlar verdiği görülmüştür.

GENETIC ALGORITHM BASED INTELLIGENT TEST PAPER GENERATION

SUMMARY

In recent years, due to the rapid development of technology, it has become possible to store and use the data effectively in an electronic environment as well as in all areas of life. The innovations provided by technology in this era, where electronic media and paperless solutions are becoming more and more widespread, make our lives easier every day. The exams that are applied for the measurement and evaluation studies which are part of the education system can be prepared in the electronic environment and can be made independent of time and place, thus offering easy access to large communities. The test questions prepared for the exams can also be stored and used in the electronic environment together with their various properties.

Along with the use of electronic question banks in the education and training, the need for the preparation of test pages in the desired characteristics emerges. In order to generate intelligent test pages, different methods and algorithms are used instead of human being, so that high quality and effective test pages can be generated at the desired level while saving time. There are studies in which traditional algorithms and methods are used for generating a test page on the desired test page from a question bank. However, the application of these methods often increases the solution time, can cause situations such as the test page being the same as the one generated in the past and may reduce the yield for test page production. Heuristic methods are often used as an alternative for problems that are difficult to solve by mathematical or traditional methods or that have a long solution time. These problems can be overcome by the heuristic optimization methods used in the artificial intelligence and optimum results can be obtained according to the desired criteria even in a question bank having limited questions. With the heuristic optimization methods, the solution can be brought to the best possible by going through the randomly selected questions without any aim of producing the best solution. The optimum result can be found for the solution by evaluating the many required criteria.

In the thesis study, a problem was solved by using an heuristic approach to the problem of generating test pages with multiple criterias and at the same time, it was aimed to reduce human labor force and time loss. The genetic algorithm adapted to the problem of the test page has been provided in a fast and efficient manner to generate the test page in the desired criteria and specifications. Web-based application software has been implemented in order to make the test page generation in this work easier for users. The genetic algorithm used in the study is compared with the standard genetic algorithm results and it is seen that the genetic algorithm used in the study gave better results.

1. GİRİŞ

Günümüz teknolojisi hızlı bir şekilde ilerlerken bununla beraber yapay zeka, makine öğrenmesi, derin öğrenme, sezisel optimizasyon alanları da gelişmekte ve daha çok hayatımıza dokunur hale gelmektedir. Yapay zeka ve makine öğrenmesi konularında geliştirilen makine ve robotlar, insanın deneyerek öğrenme yeteneklerini taklit etmekte, hatalarından ders çıkarabilmekte, insan gibi öğrenebilmekte ve vereceği kararlarda insana yakın davranabilmektedir. Bu gelişmelerle beraber teknolojiyi yakından takip eden ve teknolojiye hakim olan toplumların diğer toplumlara bir çok alanda fark atabileceği aşikardır. Teknolojiyi bilmek ve kullanmak, beraberinde gücü getirmektedir.

Teknolojinin her alanda hayatımıza girmesiyle bilgi de artık elektronik hale gelip saklanmakta ve verimli bir şekilde kullanılabilmektedir. Bilgiyi elinde tutan ve etkili bir şekilde kullanan toplumlar bu alandaki teknolojisini ileri götürebilmektedir.

Bu tez çalışmasında, yapay zekâ alanında kullanılan algoritmalardan biri olan genetik algoritma (GA) ile hatalı sonuçlardan çıkarımlar yaparak kendini iyileştiren ve en iyi çözüme kısa sürede ulaşmayı hedefleyen bir uygulama geliştirilmiştir. Çalışma ile elektronik ortamda tutulan bilgiler kullanılarak eğitim sisteminde kullanılan test sayfalarının, insan gücü kullanmadan verimli ve hızlı bir şekilde otomatik olarak oluşturulabilmesi sağlanmaktadır. Çalışma, insanın gelişiminde temel konu olan eğitim alanında kullanılan test sayfalarının oluşturulması konusunda daha etkin ve verimli bir yöntem öne sürmektedir. Çalışma ile beraber geliştirilmiş uygulama test sayfası oluşturma sürecinde insan iş gücü ve zaman kaybını oldukça azaltmakta ve çıktıların istenilen kriterlerde, verimli ve kaliteli olmasını sağlanmaktadır.

1.1 E-Öğrenme ve E-Sınav

Tüm dünyada büyük bir ilgi görerek her geçen gün daha çok yaygınlaşmakta olan e-öğrenme, bilgi teknolojilerindeki gelişmelere paralel olarak gelişip büyüyen bir eğitim şekli olmaktadır. Büyük bir hızla gelişmekte olan bilgi teknolojileri birçok

alanda olduğu gibi eğitim alanında da çeşitli araç ve metotların kullanılması için zemin hazırlamaktadır (Emir, 2006). Bilgi teknolojilerinin hazırlamış olduğu bu zeminle beraber eğitim alanında oldukça uzun bir yol alınmıştır. Geçmişten bugüne kadar ilerleyen teknolojilerle beraber internet, bilgisayarlar, telefon, tabletler ve diğer araçlar e-öğrenme alanının büyüklüğünü her geçen gün genişletmekte ve hızlı bir şekilde yaygınlaşmasını sağlamaktadır. E-öğrenme sisteminin sahip olduğu esneklik, zaman ve mekândan bağımsızlık bu sistemin gelişmesindeki en önemli faktörlerdendir. E-öğrenme sisteminin eğitim alanına getirdiği yenilikler ile beraber bu sistemin maliyetleri düşürmesi en büyük avantajlarından biri olmaktadır (Cheng ve diğ., 2009).

Geçmişte çok yaygın bir şekilde kullanılmıyor olsa da bilgi ve iletişim altyapısı konularında ilerleme sağlayan ülkemizde her geçen gün e-öğrenme teknolojilerinin, imkânlarının ve bunlara bağlı olarak kullanımının artması beklenmektedir.

Eğitim alanındaki değerlendirme işlemleri büyük ölçüde sınav sistemleriyle yapılmaktadır. Ölçme ve değerlendirme kavramları eğitim sistemiyle birlikte değer kazanan kavramlardır. Ölçme, eğitimle kazanılmış olan bilgiler, değerlendirme ise ölçme işlemine anlam kazandırma şeklinde ifade edilebilir. E-sınav, elektronik ortamda ölçme işleminin yapılması olup test tipindeki sınav sistemlerinde en yaygın şekilde kullanılan sistemdir. Test şeklindeki sınavlar diğer sınav tiplerine göre hazırlama süresi, değerlendirme süresi, rastgele soru seçimi, maliyet, saklama kolaylığı, kolay erişilebilirlik vb. birçok avantaja sahiptir (Torkul, 2004). Test şeklindeki sınav sisteminde test oluşturulmadan önce sınav zorluğu, soru sayısı, sınav süresi, geçmişte sorunun seçilme sıklığı gibi birçok parametre belirlenebilmekte olup üst seviye bilgi ve becerilerin dahi ölçülebilmesine imkân sağlamaktadır.

Günümüzde e-YDS, TOEFL, IELTS gibi İngilizce sınavları, bununla beraber Cisco, Microsoft gibi firmaların sınavları da elektronik ortamda yapılmakta ve değerlendirilmektedir (Akşam, 2014).

Bu çalışmada konu olan test sayfası üretme işlemi sonucunda belirlenen sorular, sadece elektronik ortamda değil klasik yöntemlerle de sorularak test sınavı uygulanabilir.

1.2 Akıllı Test Sayfası

Günümüz dünyasında bilgi, toplumların ekonomik seviyelerini, birbirleriyle rekabet güçlerini ve gelişmişlik seviyelerini belirleyen en önemli faktör haline gelmiştir. Bilgi bu derece önemliyken eğitim ve öğretim alanında bilgiyi sunmak, etkin bir şekilde kullanmak ve sorgulamanın da önemi her geçen gün artmaktadır. Bilgi çağında insanlar yaşları ne olursa olsun her mekânda ve her anda internetle ve çeşitli e-öğrenme metotlarıyla bilgiye erişebilmektedir. Bilgi bu derece önemliyken teknolojik gelişmelerle beraber öğrenilmiş bilgilerin ölçülmesi, değerlendirme amaçlı olarak sınavlar ve testler için etkin ve akıllı şekilde soru hazırlanması işlemleri de oldukça önemli hale gelmektedir.

Elektronik ortamda hızlı ve etkin bir şekilde hazırlanmış olan sınavlar her geçen gün artmakta ve gün geçtikçe önceden kullanılan klasik sınavların yerini almaktadır. Bilgi ve teknoloji çağında elektronik ve kâğıtsız olan çözümlerin zamanla büyük ölçüde klasik yöntemlerin yerine geçmesi kaçınılmaz bir gerçektir. Bilgilerin artık tamamen elektronik ortamda saklanacağı bu çağda, eğitim sisteminin bir parçası olan sınav sorularının da, her sorunun kendine ait çeşitli özelliklerle beraber, elektronik sistemde saklanmasını gerektirecektir. Akıllı test sayfalarının önemi de bu noktada başlamaktadır. Sorular elektronik ortama kendi özellikleriyle beraber aktarılarak elektronik soru bankaları oluşturuldukça bu soru bankalarından istenilen özelliklerde akıllı test sayfaları oluşturulabilmektedir. Bu sayede soru hazırlama bakımından zamandan kazanç olmakta, bu işlemi insan yerine hızlı bir şekilde makine yapmakta, istenilen seviyede olan test sayfasıyla, etkin ve kaliteli şekilde sınav soruları oluşturulabilmektedir. Akıllı test sayfalarının avantajları aşağıdaki gibi belirtilebilir;

- Zamandan kazanç
- Düşük maliyet
- İnsan faktörünün ve gereken iş gücünün azalması
- İnsan kaynaklı hata oranının azalması
- İstenilen seviyede kaliteli sorular oluşturulması
- Bilgi ölçümünün etkin bir şekilde yapılması
- Büyük soru bankaları için çok sorulu testleri verimli olarak hazırlama
- Güvenlik ve güvenirlilik

1.3 Test Sayfası Oluşturma Problemi

Elektronik soru bankalarında, soru bankasının büyüklüğüne, soru çeşitliliğine ve soru sayılarına göre bir konu ile ilgili sınırlı sayıda soru olabilmekte ve bu durum, soruların istenilen kriterlere bağlı olarak en uygun şekilde seçilme işlemini zorlaştırmaktadır. Soru bankası içerisinden test sayfası oluşturma işleminin geleneksel yöntemlerle çözülmeye çalışılması durumunda, test sayfası daha önce üretilen test sayfalarıyla aynı olabilmektedir. Geleneksel yöntemler, test sayfası için istenilen çok sayıda kriter söz konusu olduğunda istenilen çözüm için daha uzun süre alabilir ya da belirli bir sürede iyi bir çözüme ulaşamayabilirler. Yapılan çalışmalarda select-random, backtracking algoritmaları gibi algoritmalar kullanılmış olsa da bu algoritmaların uzun süre alması ve istenilen kriterlere uygunluğu konusunda yeteri kadar başarılı olamaması gibi dezavantajları olmuştur (Jun, 2014; Zhang ve Zhu, 2015; Zhong ve Wang, 2010). Sezgisel optimizasyon yöntemleri kullanılarak test sayfası oluşturma problemi çözüldüğünde ise çok kısıtlı sorulara sahip olan bir soru bankasında bile kısa süre içerisinde istenilen kriterlere göre en iyi veya en iyiye yakın çözümler elde edilebilmektedir. Sezgisel optimizasyon yöntemleriyle her zaman en iyi çözümü üretme gibi bir amaç güdülmeden, rastgele seçilen sorular üzerinden gidilerek çözüm en iyiye yakınlaştırılabilmektedir. İstenilen çok sayıda kriter değerlendirilerek çözüm için kabul edilebilir seviyede sonuçlar bulunabilmektedir.

1.4 Literatür Araştırması

Akşam (2014) tarafından gerçekleştirilen yüksek lisans tez çalışmasında, soru bankası üzerinden test sayfası oluşturulmuştur. Geliştirilen web uygulamasında sorular zorluk seviyeleri girilerek hazırlanabilmektedir. Test sayfası oluşturulurken amaç fonksiyonunda kriter olarak yalnızca sorunun zorluk seviyesi yer almakta olup problemin çözümü için sezgisel yöntemlerden biri olan parçacık sürü optimizasyon (PSO) yöntemi tercih edilmiştir.

Karataş (2009) tarafından gerçekleştirilen yüksek lisans tez çalışmasında, akıllı bir e-soru sınav sistemi tasarımı ve uygulaması yapılmıştır. Buna göre dil işleme teknikleri kullanılarak sözcüklerden soru cümleleri elde edilebilmektedir. Ders içeriği ile ilgili metinlerin sisteme girilmesiyle beraber o metinlerle ilgili sorular oluşturulmakta, bu

sayede öğrencinin konu hakkındaki bilgisini ve eksiklerini anlaması hedeflenmektedir. Doğal dil çözümleme yöntemlerinin ve cümle öğelerinin çözümleme algoritmasının kullanıldığı bu çalışmada üretilmiş olan anlamsız soruları öğretmenin hazırlama aşamasında elemesi beklenmektedir.

Beyazşekeroğlu (2015) tarafından gerçekleştirilen yüksek lisans tez çalışmasında, Moodle öğrenme yönetim sistemleri üzerinde 240 soruluk bir soru bankası içinden GA kullanılarak test soruları hazırlanmıştır. Yapılan bu çalışmada eğitmen, soru sayısı ve sınavın ortalama zorluk seviyesini girmekte ve amaç fonksiyonunda da bu değerler kullanılarak test sayfası oluşturma problemine çözüm aranmaktadır. Ayrıca amaç fonksiyonunda soruların geçmiş sınavda seçilme durumları da ihlal kısıtlaması olarak değerlendirilmiştir.

Yıldırım (2008) tarafından gerçekleştirilen çalışmada, bir soru bankası üzerinde GA ile test sayfası oluşturma problemine çözüm aranmıştır. Bu çalışmada, standart GA'nın test sayfası oluşturma problemi üzerinde doğrudan kullanılamayacağı, testte aynı soruların oluşabileceği ifade edilmiştir. Çalışmada tekrarlı soruları önleyen bir mutasyon işlemi önerilmiş, yapılan analiz ve testlerde farklı zorluk seviyeleri ve farklı soru sayıları için önerilen algoritmanın başarı değerleri incelenmiştir. Test sayfası oluşturma probleminin çözümü için amaç fonksiyonu içinde soruların zorluk seviyeleri ve geçmişteki sorulma sıklığı bilgileri kullanılmıştır.

Zhong ve Wang (2010) tarafından gerçekleştirilen çalışmada, test sayfası oluşturma problemininin GA ile çözümü üzerinde durulmuştur. Çalışmada GA için uygulanan farklı çaprazlama ve mutasyon işlemlerinin, çok kriterli test sayfası oluşturma problemi üzerindeki verimi gösterilmiştir. Çaprazlama işleminde geleneksel yöntemden farklı olarak bir olasılık formülü kullanılmıştır. Soru sayısı, soru tipi, toplam puan ve süre gibi kriterler kullanılarak amaç fonksiyonunda sorunun zorluk derecesinin dikkate alındığı belirtilmiştir.

Nguyen ve diğ. (2011) tarafından gerçekleştirilen çalışmada, test sayfası oluşturma probleminin çözümü için sezgisel algoritmalardan daha farklı bir metot önerilmiştir. Çalışmada kısıt tabanlı böl ve yönet tekniği (Constraint-based Divide-and-Conquer technique) önerilmekte olup bu teknik GA, karınca koloni algoritması (KKA), PSO, tabu arama algoritması gibi farklı sezgisel metotlarla karşılaştırılmış ve dört farklı veri seti için sonuçlar değerlendirilmiştir. Önerilen algoritmanın, zaman konusunda,

kısıtlara uyum konusunda, kalite ve farklılık konularında diğer metotlara göre daha performanslı olduğu test sonuçlarına dayanan grafiklerle gösterilmiştir. Çalışmada önerilen algoritmanın test sayfası probleminin çözümü için sezgisel yöntemlere göre daha başarılı olduğu öne sürülmüştür.

Bhirangi ve Bhoir (2016) tarafından gerçekleştirilen çalışmada, test sayfası oluşturma problemine rastsal bir algoritma yaklaşımı sunulmuştur. Rastsallık esasına dayanan shuffling algoritmasının kullanıldığı bu çalışmada seçilen sorular işaretlenerek tekrar seçilmelerinin önüne geçilmiştir. Rastgele bir yaklaşımla beraber soruların tekrarsız şekilde olması üzerinde durulan çalışmada, sistemi kullanacak kişiler için rol bazlı bir yetkilendirme işlemleri yapılarak soruların güvenliğinin, kaynaklara erişimin kontrol altında tutulduğu belirtilmiş olup bununla ilgili Java platformunda bir uygulama geliştirilmiştir.

Yong-kang ve Wang-ren (2011) tarafından gerçekleştirilen çalışmada, otomatik test sayfası üretimi konusuna çözüm ararken aralık bulanık teorisi (interval fuzzy theory) kullanılmış ve test sayfasının zorluk seviyesinin kapsamlı bir şekilde değerlendirildiği belirtilmiştir. Çalışmada kriter olarak zorluk seviyesi ve bilgi puanı ele alınmıştır. Test sayfasının zorluk derecesi değerlendirilirken bulanık mantık teorisi ile sayfalar değerlendirilmekte ve bu yaklaşım öğrencilerin cevaplarıyla beraber kombine edilmektedir. Sınavlarda kullanılabilecek çevrimiçi bir sistem üzerinde çalışabilen ve otomatik olarak test sayfası üretebilen bir yaklaşım geliştirilmiştir.

Hairui ve Hua (2008) tarafından gerçekleştirilen çalışmada, test sayfası oluşturma problemine çözüm aranırken öğretmen, test sayfası, internet, sınav bazlı sosyal kavramlar üzerinde çoklu faktör tabanlı bir yaklaşım kullanılmıştır. Algoritma içinde rastgele seçim (random selection) ve yaklaşık eşleştirme (approximate match) yöntemleri uygulanmıştır.

Li ve diğ. (2016) tarafından gerçekleştirilen çalışmada, nesne tabanlı programlarda test veri üretiminin doğruluğu ve test verisi üretirken uyulması gereken yöntemlerin sırasını öngörecek test zinciri kavramı sunulmuştur. Çalışmada nesne tabanlı programların test veri üretimi için bir çerçeve çizilmiştir. Parametre listesinin güncellenmesi için PSO algoritmasını ve metot sırası güncellemesi için GA'yı kullanmışlardır. Çalışmada test verilerini kullanarak başlangıç nüfusunu iyileştirmek

için bir yöntem sunulmuştur. Sonuçların önerilen yöntemin nesne tabanlı programlar için test verileri üretmede başarı oranını ve verimliliği arttırdığı ifade edilmiştir.

Ming-Zhu ve diğ. (2013) tarafından gerçekleştirilen ve otomatik test sayfası üretiminin sınav sistemindeki öneminin vurgulandığı çalışmada, geliştirilmiş bir GA yaklaşımı önerilmektedir. Çalışmada rastgele iterasyon yöntemi, standart GA ve önerilen GA üzerinde deneysel sonuçlar karşılaştırılmış olup önerilen GA'nın diğerlerine göre zaman ve hata oranı bakımından daha iyi olduğu belirtilmiştir. Soru sayısı, soru tipleri ve puan gibi bilgiler isteğe göre belirlenerek amaç fonksiyonunda ise zorluk derecesi, bilgi puanı gibi faktörlerin göz önüne alındığı ifade edilmiştir.

Jia ve diğ. (2011) tarafından gerçekleştirilen çalışmada, bilgisayar ağındaki test sistemlerinde otomatik test sayfası oluşturma işleminin önemi anlatılarak geliştirilmiş bir GA ile probleme çözüm arandığı ifade edilmiştir. Kullanılan soru bankası için soru tipine göre her bölüme ait soru sayısı, ortalama zorluk, ortalama farklılık, ortalama bilgi puanı gibi bilgileri gösterilmiştir. Çalışmanın sonuçları, önerilen algoritma, standart GA ve geri izleme (backtracking) algoritmalarıyla karşılaştırılmıştır.

Xiumin ve diğ. (2011) tarafından gerçekleştirilen çalışmada, GA kullanılarak akıllı test sayfası üretme problemine çözüm arandığı ifade edilmiştir. Çalışmada GA için kullanılacak olan başlangıç nüfusu rastgele değil, optimize edilip iyileştirilmiş olarak oluşturulmuştur. Duruma göre kendini adapte eden çaprazlama ve mutasyon işlemleri kullanılmıştır. Bu yönteme göre önceki nüfusun ve yeni nüfusun ortalama uygunluk değerleri karşılaştırılarak bunun sonuca göre mutasyon ve çaprazlama işlemlerinin uygulanıp uygulanmayacağına karar verilmektedir. Adaptif yöntem ile standart yöntem, rastgele oluşturulan nüfus ile iyileştirilmeyle oluşturulan nüfus gibi farklı yöntemler test edilerek, sonuçlar gösterilmiştir. Önerilen yöntemlerin standart yöntemlere göre daha başarılı sonuçlar verdiği açıklanmıştır.

Jun (2014) tarafından gerçekleştirilen çalışmada, geliştirilmiş bir GA ile test sayfası oluşturma probleminin çözüldüğü ifade edilmiştir. Çalışmada bölüm bazlı bir çaprazlama işlemi uygulanmıştır. Çaprazlama işleminde iki birey arasındaki sorular çaprazlanırken aynı bölümlerde olması gerekmektedir ve her bir bölüme ait sorular kendi aralarında çaprazlanmaktadır. Mutasyon ve çaprazlama işlemleri uygulanırken belirli bir olasılık formülüne göre işlemlerin yapılmasına karar verilmiştir. Test

sayfasının mevcut özellikleri göz önüne alınarak çaprazlama işleminin uygulanmasına karar verilmiş ve bu şekilde adaptif bir yöntem kullanılmıştır. Çalışmada kullanılan amaç fonksiyonu içinde teste ait puan, bilgi puanı, zorluk derecesi, farklılık puanı gibi faktörler bulunmaktadır. Çalışmada önerilmekte olan yöntemin standart GA'ya göre daha iyi sonuçlara sahip olduğu testler sonucunda elde edilen grafiklerle gösterilmiştir.

Zhang ve Zhu (2015) tarafından gerçekleştirilen çalışmada, GA ile otomatik test sayfası oluşturma problemi için çözüm geliştirildiği belirtilmiştir. Çalışmada farklı olarak, bölüm bazlı bir çaprazlama işlemi olmayıp soru tipine göre çaprazlama işlemi uygulanmıştır. Çaprazlama işleminde iki birey arasındaki sorular çaprazlanırken aynı soru tiplerinde olmaları kuralı uygulanmıştır ve her bir soru tipine ait sorular kendi aralarında çaprazlanmaktadır. Amaç fonksiyonunda kriter olarak zorluk seviyesi ve puan kriterleri kullanılmıştır. Elde edilen sonuçlar, süreye göre, iterasyon sayısına göre, soru tiplerine göre karşılaştırılmış ve standart GA'ya göre önerilen algoritmanın daha iyi sonuçlar verdiği ifade edilmiştir.

Sun (2009) tarafından gerçekleştirilen çalışmada, test sayfası oluşturma problemi için PSO yönteminden geliştirilmiş olan ayrık PSO yöntemi önerilmektedir. Önerilen yöntemde hata oranları ile ağırlıkların çarpımı amaç fonksiyonunu oluşturmaktadır. Yapılan çalışmada çaprazlama ve mutasyon işlemleri de kullanılmıştır.

Shan (2010) tarafından gerçekleştirilen çalışmada, GA üzerinde faklı bir strateji kullanılarak, çoklu iş parçacığı (multi-threaded) yaklaşımıyla test sayfası üretme problemini daha hızlı şekilde çözeceği öne sürülen bir çözüm geliştirildiği ifade edilmiştir. Standart GA ile önerilen algoritma karşılaştırılarak, önerilen algoritmanın daha iyi sürelerde ve daha etkin bir şekilde problemi çözdüğü belirtilmiştir.

Xiong ve Shi (2010) tarafından gerçekleştirilen çalışmada, GA ile test sayfası oluşturma problemine çözüm aranırken matematiksel model üzerinde durulmuştur. Çalışmada standart GA üzerinde çaprazlama ve mutasyon işlemlerinde iyileştirmeler yapılmış olup parametrelerin girildiği ve sonuçların gösterildiği bir uygulama da gerçekleştirilmiştir. Çaprazlama ve mutasyon işlemleri belirlenmiş bir olasılık formülüne göre gerçekleşmektedir. Çaprazlama işleminde standart GA'ya göre farklı bir yaklaşımla çocuk bireylerin elendiği bir yapı tasarlanmıştır. Buna göre çaprazlama işleminden sonra oluşan çocuk bireylerin uygunluk değerleri

hesaplanarak ebeveyn bireyleriyle karşılaştırılmış ve bunun sonucunda çocuk bireylerin uygunluk değerleri ebeveynlerden daha kötü ya da eşit ise bu şartı sağlayan çocuk bireyler elenmiştir. Bu şekilde nüfusun daha iyi olacağı öne sürülmüştür.

2. OPTİMİZASYON VE SEZGİSEL ALGORİTMALAR

2.1 Optimizasyon

Bir problemin çözümünde, belirli koşullar altında olabilecek bütün alternatifler arasından en iyisinin seçilme işlemine optimizasyon denir. Farklı bir ifadeyle optimizasyon, soruna en uygun çözümün bulunmasıdır. Karmaşıklık derecesi fazla olan ve karar verme işlemlerinin gerektiği problemler optimizasyon problemleridir.

Bilgisayar bilimlerinin de dahil olduğu çeşitli bilim ve mühendislik problemlerinde optimizasyon amacıyla birden fazla mevcut çözüm arasından en iyisi aranır. Optimizasyon problemlerinin çözümü için amaçlanan işlem fonksiyonun değerini minimum ya da maksimum yapmaya çalışmaktır. Bir fabrikanın verimliliğini arttırmak için üretimini arttırması maksimuma optimizasyon, üretim süresini azalmak ise minimuma optimizasyon yapmak olarak görülür. Matematiksel optimizasyon, problemlerin çözümü için matematiksel bir formüle dayanarak en iyi sonucu bulan yöntemlerdir. Bu yöntemlerde belirsizlik, rastgelelik, tahmin ve olasılığa dayalı işlemler yer almaz. Deterministik yöntemlerde algoritma kaç defa çalıştırılırsa çalıştırılsın aynı sonuç üretilmektedir (Sel, 2013).

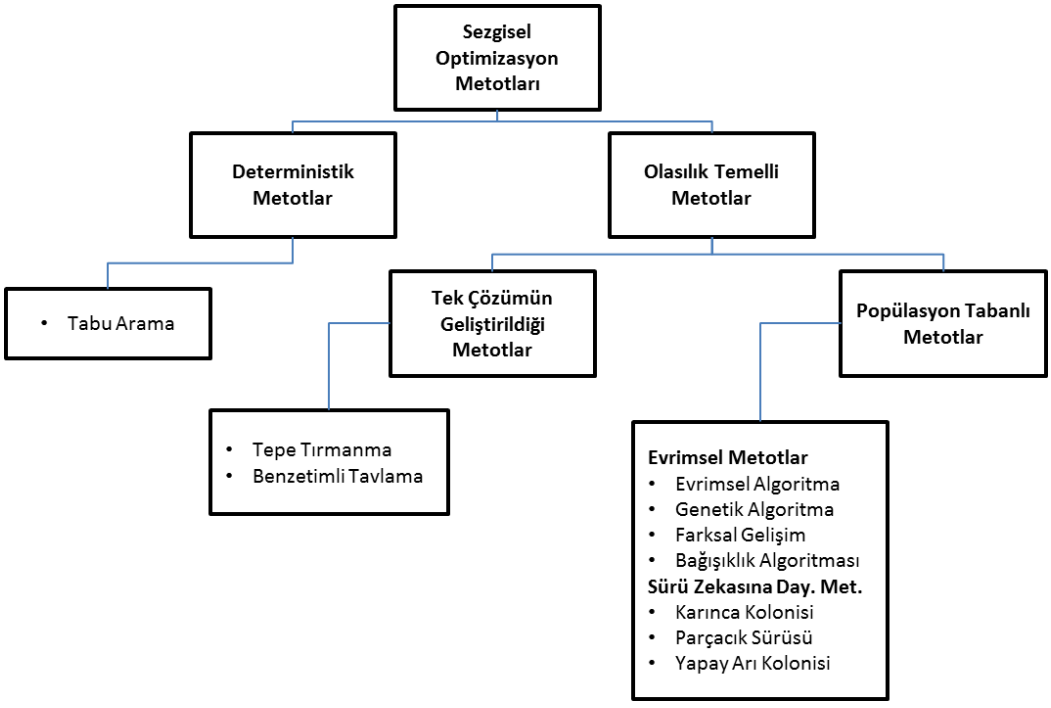
2.2 Sezgisel Algoritmalar

Sezgisel algoritmalar, karmaşık ve büyük boyutlu optimizasyon problemlerinin çözülmesinde optimum sonuca yakın makul sonuçlar vermeyi hedefleyen yöntemlerdir. Sezgisel algoritmalar aslında günlük hayatımızda sürekli kullanıyor olduğumuz algoritmalardır. Örneğin bir yere giderken bulunduğumuz konuma göre yön duygularımıza güvenerek yolun bizi nereye götüreceğini bilmeden hareket etmek ve yol ayrımlarında sezgilerimizi kullanarak yönümüzü belirlemek böyle bir yaklaşımdır.

Bilgisayar bilimleri ve yapay zekâ alanında problem çözme tekniklerinden biri olan sezgisel algoritmalar kullanılarak bulunan sonuçların en iyi olup olmadığı

önemsenmez, bunun yerine en iyi sonuca yakın sonuçlar da kabul edilebilir. En iyi sonucu garanti etmeyen bu algoritmalar makul bir sürede en iyi sonuca yakın bir çözümü garanti ederler. Bu yapılarıyla beraber en iyiye yakın sonuca hızlı ve kolay bir şekilde ulaşmayı hedeflemektedirler. Sezgisel algoritmalar iyi bir sonuca ne kadar kısa sürede yaklaşıyorlarsa o kadar etkili oldukları kabul edilebilir. Bununla beraber sezgisel algoritmalar, problemi makul bir zamanda çözerken, bir problemi her defasında aynı sürede çözeceklerini de garanti etmezler. Sezgisel algoritmalar problemin kesin bir çözüme sahip olmadığı ve problemin birçok parametre ve kısıtlara bağlı olması durumunda, öğrenme amaçlı olarak tercih edilebilirler.

Genellikle doğadan esinlenerek geliştirilmiş olan sezgisel algoritmalar pek çok problemin çözümünde başarılı sonuçlar elde etmişlerdir. Özellikle büyük boyutlu veya karmaşık optimizasyon poblemlerinde kabul edilebilir sürelerde en iyiye yakın çözümler bulabildiklerinden dolayı son yıllarda sıkça tercih edilmektedir. Problemlerin kesin tek bir çözüme sahip olmadığı durumlar yine sezgisel algoritmaların tercih edilme sebeplerindendir. Literatürde başarılı bir şekilde problemlere çözüm üretebilen sezgisel algoritmalara örnek olarak; genetik algoritma, parçacık sürü optimizasyonu, karınca kolonisi algoritması, tabu arama, benzetimli tavlama, yapay arı kolonisi algoritması gibi pekçok yöntem verilebilir. Sezgisel optimizasyon metotları Şekil 2.1 olduğu gibi sınıflandırılabilir (Akay, 2009).



Şekil 2.1 : Sezgisel algoritmaların sınıflandırılması.

2.2.1 Tepe tırmanma algoritması

İteratif yerel arama algoritmalarından biridir. Bir grafikte bulunan maksimum değere sahip tepe noktasının aranması işlemine tek bir noktadan başlanır (Ackley, 1987). Bu grafikte bulunan noktalar aranırken yapılan hareket tepe tırmanmaya benzetilmiş ve algoritma adını bu şekilde almıştır. Algoritmadaki amaç başlangıç noktası olarak belirlenen bir noktadan komşu noktalara bakarak daha iyi bir sonucu aramaktır. Her iterasyon ile mevcut çözümün komşusundaki çözümlere bakılır ve komşu çözümlerde mevcut çözümden daha iyi bir çözüm varsa yeni çözüm olarak seçilir. Komşu çözümlerin olası durumları aşağıda belirtilmiştir;

- Mevcut noktanın bir tarafında çözüm iyileşirken diğer tarafında çözüm kötüleşebilir.
- Mevcut noktanın iki tarafındaki komşularda da çözüm kötüleşebilir. Bu durumda algoritma bulunduğu noktayı tepe noktası sanarak yerel optimum değerde takılmış olur ve algoritma bu noktada kalarak daha iyi sonuçları bulamaz.
- Mevcut noktanın iki komşusunda da çözüm iyileşiyor olabilir. Bu durumda bulunulan nokta çözüm için kötü sonuçlardan biri olabilir.

Tepe tırmanma algoritması sezgisel arama algoritmaları arasında en iyisi değildir ancak basit yapısı, tasarım kolaylığı ve çözüme hızlı ulaşma gibi avatajlarından dolayı basit optimizasyon problemlerinde tercih edilmektedir. Klasik tepe tırmanma algoritmasından farklı olarak iki yöne tırmanan, rastgele komşu seçilen, rastgele tekrar başlamalı algoritmalar gibi tepe tırmanmanın farklı şekillerde geliştirildiği algoritmalar da mevcuttur.

2.2.2 Tabu arama algoritması

Tabu arama, Glover tarafından geliştirilmiş iteratif bir arama algoritmasıdır. Tepe tırmanma algoritmasında istenmeyen bir durum olan yerel optimum değerlere takılmayarak genel optimum değeri bulmaya çalışır (Glover, 1989). Hafıza olarak bir tabu listesi kullanır ve her iterasyonda en iyi komşu çözümü bir değerlendirme fonksiyonu kullanarak bulur. Her iterasyonda gerçekleştirilen hareket tabu listesinde tutulur. Karşılaşılan durumlar hakkında uygulayan tarafından belirlenmiş olan bilgiler belirli bir uzunlukta olacak şekilde bu listede tutulmaktadır. Bu sayede arama işleminin sonsuz döngü olan çember hareketini yapması engellenir ve çözüme daha

kolay ulaşmak hedeflenir. Başlangıç çözümü rastgele ya da bir algoritmaya göre seçilebilir. Yeni çözüm oluşturma fonksiyonu problemin türüne göre değişebilir ve sonucu doğrudan etkiler.

Tabu arama metodunda önceki iterasyonlarda karşılaşılan durumlar hafızada tutulur ve belirli bir stratejiye göre uygulanacak seçimlerle daha iyi sonuçlara ulaşılmaya çalışılır. Bu stratejiye göre belirlenmiş kötü bir seçim rastgele olarak belirlenmiş bir seçimden daha iyi olabilmektedir. Hafızanın kullanıldığı akıllı bir sistemde belirlenmiş olan stratejiye göre yapılacak kötü bir seçim bu stratejinin iyileştirilmesi için yol gösterici olabilmektedir (Glover ve Laguna, 1997).

Tabu arama algoritmasında kısa ve uzun süreli hafıza tutulabilir. Tabu listesi kısa süreli hafıza olarak değerlendirilir, arama sırasında yapılan hareketler sınırlı sayılı olan bu tabu listesinde tutulur. Belirli bir süre sonunda algoritmanın sabit bir çözüme bağlı kalmasını önlemek amacıyla tabular yıkılarak kayıtlar tabu listesinden çıkarılır. Tabu listesinden kısıtları çıkarma işlemi genellikle FIFO (ilk giren ilk çıkar) stratejisi ile yapılmaktadır. Uzun süreli hafıza ile arama işlemi çözüm uzayında şimdiye kadar arama yapılmamış yeni bölgelere yönlendirilir (Reeves, 1993). İyi çözümlerin bulunma olasılığı tabu listesi ile beraber uzun süreli hafıza kullanımıyla beraber güçlendirilir. Araştırmanın sadece belirli dönemi değil bütünü ile ilgili bilgiler bu hafıza türünde tutulabilir (Gülcü, 2006). Algoritma daha önceden belirlenmiş olan maksimum iterasyon sayısına ulaşınca ya da bulunan çözümün yeterlilik şartını sağlaması durumunda sonlandırılır.

2.2.3 Benzetimli tavlama algoritması

1983 yılında Kirkpatrick ve arkadaşları tarafından önerilmiş olan bu algoritma metal gibi katı maddelerin ısıtılması ve sonrasında yavaş yavaş soğutulması işlemine benzetilmiş ve adını buradan almıştır (Kirkpatrick, 1983). Katı malzemenin şekil alması ve malzemeyi işlemenin kolaylaşması amacıyla ona uygulanan ısıl işlemlere genel olarak tavlama adı verilir. Bu işlemde, belirli bir sıcaklığa ulaşıncaya kadar ısıtılan malzeme sıcaklığı maksimum dereceye geldikten belirli bir süre sonra soğutulma işlemine başlanır. Bu soğuma işleminin doğru şekilde uygulanması durumunda malzemeden beklenilen sonuçlar alınabilir.

Yerel optimum değerlerinden kurtulmak için daha kötü olan komşu çözümler de küçük bir ihtimalle de olsa kabul edilebilir. Böylece genel optimum noktasına ulaşılabilir.

Algoritmada yerel optimum değerlerden korunmak için komşu çözümler bazı durumlarda kabul edilmekte ve bu şekilde genel optimum değeri aranmaya çalışılmaktadır. Komşu çözümlerin kabul edilmesini sağlayan olasılık değeri ise sıcaklık değerine bağlıdır. Algoritmada sıcaklık yüksek olursa olasılık değeri de yüksek olmakta, sıcaklık düşük olduğunda ise olasılık değeri azalmaktadır, bundan dolayı sıcaklık değerinin uygun bir seviye ile başlatılması gerekmektedir.

Algoritma özellikle lineer bir modele sahip olmayan ve kombinasyonel problemlerin çözümünde kullanılmaktadır.

2.2.4 Karınca koloni algoritması

Doğadaki canlılardan esinlenen karınca koloni algoritması (KKA), optimizasyon problemlerinin çözümleri için etkin bir şekilde kullanılan popülasyon tabanlı algoritmalardan biridir (Dorigo, 1997). Bugüne kadar birçok KKA geliştirilmiş olup bunlardan ilki Dorigo ve arkadaşları tarafından geliştirilerek Gezgin Satıcı Problemi (Travelling Salesman Problem) üzerinde uygulanmıştır (Maniezzo, 2004).

Doğadaki gerçek karınca kolonileri incelendiğinde karıncaların birbirleri arasındaki bağı koruyan bir feromon maddesi olduğu görülmüştür. Feromon, karıncaların yönlerini bulmasını, birbirleriyle iletişim kurmasını sağlayan karıncaların üretmiş olduğu doğal bir salgıdır. Karıncaların geçtikleri yolda feromon salgısının fazla olması o yolun karıncalar tarafından daha yoğun şekilde kullanıldığını göstermekte ve buna bağlı olarak bu yolun seçilme olasılığını arttırmaktadır. KKA'da bu durumdan esinlenilir ve sanal karıncalar kullanılarak mesafelerin belirli olduğu bir model üzerinden en kısa yolu bulma problemine çözüm aranır. Birim zamanda kısa olan yoldan geçen karıncaların miktarı uzun yoldan giden karıncaların miktarına göre daha fazla olacak ve bu da kısa yolda uzun yola göre feromon miktarının daha fazla olmasına sebep olacaktır. Kısa olan yolda fazla, uzun olan yolda ise az feromon birikir ve bundan dolayı feromon miktarı ile yol uzunluğu arasında ters orantılı bir ilişki olur (Url-2).

3. GENETİK ALGORİTMA

Genetik Algoritma (GA), doğadaki seçim ilkelerini temel alarak çalışan arama ve optimizasyon yöntemlerinden biridir. GA, evrim teorisinden esinlenerek geliştirilmiş olup biyolojik evrimin işleyiş sürecini taklit eden bir algoritmadır. Bu yapısıyla da yapay zekânın hızla gelişmekte olan alanlarından biri olarak kabul edilmektedir. Mevcut çözüm, algoritmanın çalışmasıyla sürekli olarak daha da iyileştirilmeye çalışılır. Algoritma, çaprazlama, mutasyon ve seçim işlemleri olmak üzere genetikteki üç temel biyolojik süreç üzerine kurulmuştur.

GA'yı bugünkü yapısından çok farklı olsa da ilk olarak Bagley, Rosenberg, De Jong gibi isimler kendi çalışmalarından kullanmışlardır. Bagley 1967 yılında bir oyun programını yenmek üzerinde çalışarak GA'nın bugüne ait temellerini atmıştır (Özkan, 2003). Aynı dönemlerde Rosenberg'de yaptığı çalışmasında GA'ya biyolojik faktörleri dahil etmiştir. De Jong ise algoritmaya fonksiyon minimum değerini ekleyerek matematiksel olarak katkı sağlamıştır (Goldberg, 1989).

GA ilk olarak Michigan Üniversitesinde psikoloji ve bilgisayar bilimi uzmanı olan John Holland tarafından literatüre kazandırılmıştır (Goldberg, 1989). Holland'ın 1975 yılında yayınlanan "Adaptation in Natural and Artificial Systems" adlı kitabında GA biyolojik sistemlerin soyut bir hali olarak ifade edilmiştir (Holland, 1992).

Holland'ın doktora öğrencisi ve aynı zamanda bir inşaat mühendisi olan Goldberg bayrağı Holland'dan devralarak GA'yı pratikte daha ilerilere taşımıştır. Goldberg'in yayınlanan kitabına kadar GA'nın gerçek hayatta kullanımı mümkün olmayan ve fazla yararı olmayan bir araştırma konusu olduğu düşünülmüştür. Ancak Goldberg, yazdığı kitabında (Goldberg, 1989) GA'nın kullanılabileceği 83 farklı uygulamayı sunarak, GA'nın pratikte yararlı bir araştırma konusu olduğunu göstermiştir. Öyle ki Goldberg'in gaz boru hatlarının denetimi üzerine yaptığı doktora tezi doktora tezi (Goldberg, 1983) ona 1985 yılında National Science Foundation genç araştırmacı ödülünü kazandırmıştır.

Algoritma, genetikten esinlenen bir yapıda doğal seçim, çaprazlama, mutasyon işlemleriyle bir sonraki nüfusun (popülasyon) iyileştirilmesini hedefler. GA, kromozomlardan oluşan bir nüfusa sahiptir. Her bir kromozom en küçük yapı taşı olan genlerden oluşur. Seçim işleminde, nüfus içinde yer alan bu koromozomlardan çiftleşme yapılacak olanlar belirlenir. Çaprazlama işleminde, seçilmiş olan iki kromozomun genlerinin karşılıklı olarak değiştirilmesi ile yeni kromozomlar oluşturulur. Mutasyon işlemi, rastgele olarak kromozomlardaki bazı genlerin değiştirilmesiyle yapılır. Bu yaklaşım, seçime daha uygun kromozomların çiftleşmesi sonucu oluşacak olan yeni nesillerin daha kaliteli ve uygun olacağı temeline dayanır. Holland'ın keşfetmiş olduğu nüfus temelli çaprazlama, mutasyon gibi metotlar bu alanda büyük bir yenilik olmuştur.

GA başlangıç çözümünden bağımsızdır ve çözüm adayları üzerinden paralel olarak arama yapar. Nüfus adı verilen ve içinde rastgele oluşturulmuş adayların oluşturduğu bir çözüm kümesiyle algoritma başlatılır. Bu nüfus, genetikte kullanılan çaprazlama, mutasyon ve seçim gibi temel metotlar ile iyileştirilirken, yeni oluşacak olan nüfusun öncekine göre daha iyi olacağı beklenir. Nüfusun iyileştirilme süreci en iyi çözüm bulunana kadar ya da en başta belirlenmiş olan bir döngü limiti tamamlanıncaya kadar devam eder.

GA, mühendislik, finans, pazarlama, üretim, çizelgeleme, yerleşim, sistem güvenilirliği, taşıma, araç rotası belirleme, makine öğrenmesi, yapay zeka gibi bir çok alanda optimizasyon problemlerinin çözümü için kullanılmaktadır. GA'da kullanılan temel terimler aşağıda verilmiştir.

3.1 Gen

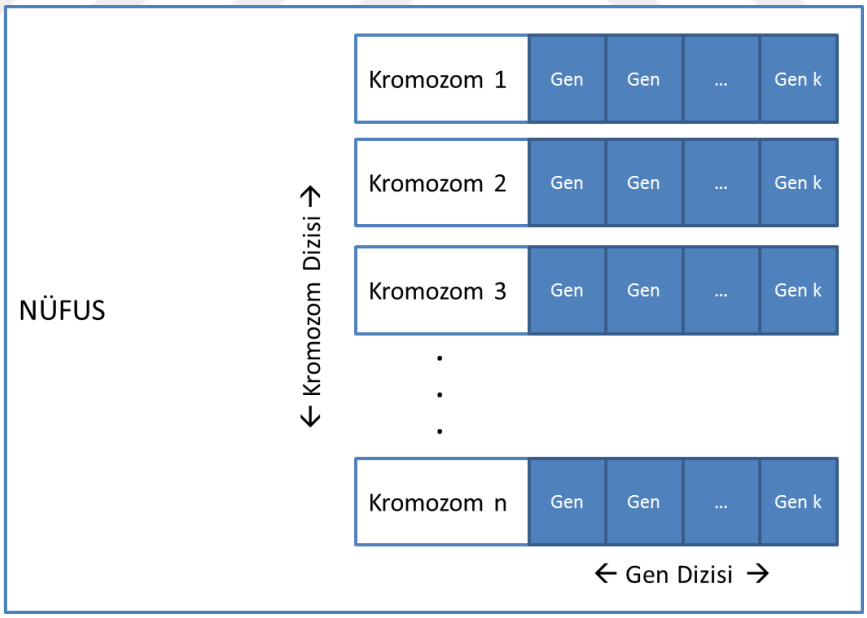
Bireyin kalıtsal özelliklerini taşıyan kalıtımın en temel ünitesidir. Gen, genetik unsurun en küçük yapı taşıdır. Kalıtsal özellik taşıyan bu genlerin bir araya gelmesiyle tüm bilgileri taşıyan kromozomlar oluşur. GA'da hedef problemin aday çözümlerini oluşturan bir kromozomdaki anlamlı en küçük bilgi gen olarak ifade edilmektedir.

3.2 Kromozom (birey)

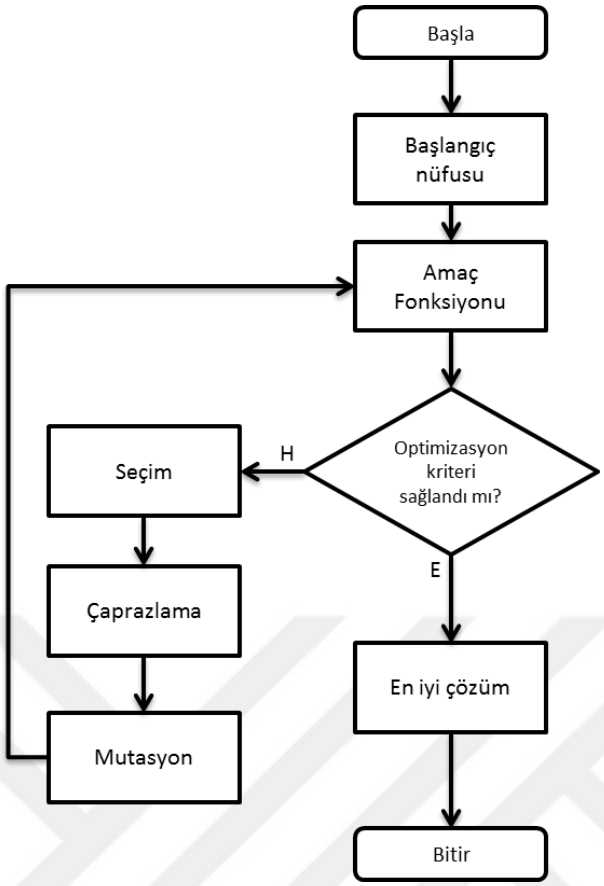
Genlerden oluşan dizidir. Birey olarak insan düşünülecek olursa; insan vücudunda trilyonlarca hücre bulunmaktadır. Her bir hücrenin çekirdeklerinde ise 23 çift kromozom bulunur. Bu kromozomlar ise ebeveyn olan anne ve babadan gelmektedir. GA'da kromozom nüfus içindeki bireye karşılık gelmekte olup aynı zamanda hedef problemin adayı çözümünü temsil etmektedir.

3.3 Nüfus (popülasyon)

Kromozomlardan oluşan bir topluluktur. Nüfus içerisindeki her bir birey, problemin olası bir çözümünü temsil etmektedir. Nüfus her jenerasyonda daha iyi bireylerden oluşmaya çalışacağı için zamanla değişip iyileşmektedir. Probleme göre nüfusta yer alan kromozom sayısının fazla olması, daha fazla hesaplama gerektirerek çözüme giden zamanı arttırmakta, kromozom sayısının sayının az olması ise nüfus içindeki çeşitliliği yok edebilmektedir. Nüfus içinde yer alacak kromozom sayısı problemin yapısına uygun olarak belirlenebilir. Nüfus, kromozom ve genler arasındaki ilişki Şekil 3.1'de, GA'nın akış diyagramı Şekil 3.2'de gösterilmektedir.



Şekil 3.1 : Nüfusta kromozom ve genlerin görünümü.



Şekil 3.2 : Genetik algoritma akış diyagramı.

3.4 Kodlama

GA'nın önemli bir kısmı olan kodlama, problemin çözümü için en başta belirlenmesi gereken yapılardandır. Oluşturulacak olan GA'nın hızlı ve sağlıklı çalışması için kodlama doğru tercih edilerek yapılmalıdır. Probleme göre uygulanabilecek farklı kodlama türleri vardır.

3.4.1 İkili kodlama

Bu kodlama türünde her kromozom 0 ve 1'lerden oluşan bit dizisidir. Bu bit disizindeki her bit çözümün belli bir karakteristik özelliğini taşır. Bit dizisi ise bir sayıyı temsil etmektedir (Nabiyev, 2005). Bu kodlama türü, arama uzayını bazı durumlarda istenilen şekilde temsil edememektedir.

3.4.2 Değer kodlama

Bu kodlama türünde her kromozom değerlerden oluşan bir dizidir. Reel sayılar gibi karmaşık değerlerin kullanıldığı problemlerin çözümünde ikili kodlama yerine tercih

edilmektedir. Bu kodlama yönteminde kromozom, reel sayıların oluşturduğu bir vektör şeklinde kodlanmaktadır. Bu yöntemle çok sayıda karar değişkeninin yer aldığı büyük vektörlerin temsili mümkündür (Erdal, 2007).

3.4.3 Permutasyon kodlama

Permutasyon kodlama yönteminde kromozomlar numaralar dizisinden oluşmaktadır (Dilaver, 2015). Bu kodlama türünde kromozomlar sıradaki konumu belirten numerik karakterlerden oluşturulabilir. Daha çok gezgin satıcı, çizelgeleme, görev sıralama gibi sıralamanın önemli olduğu problemlerde kullanılır.

3.5 Seçim

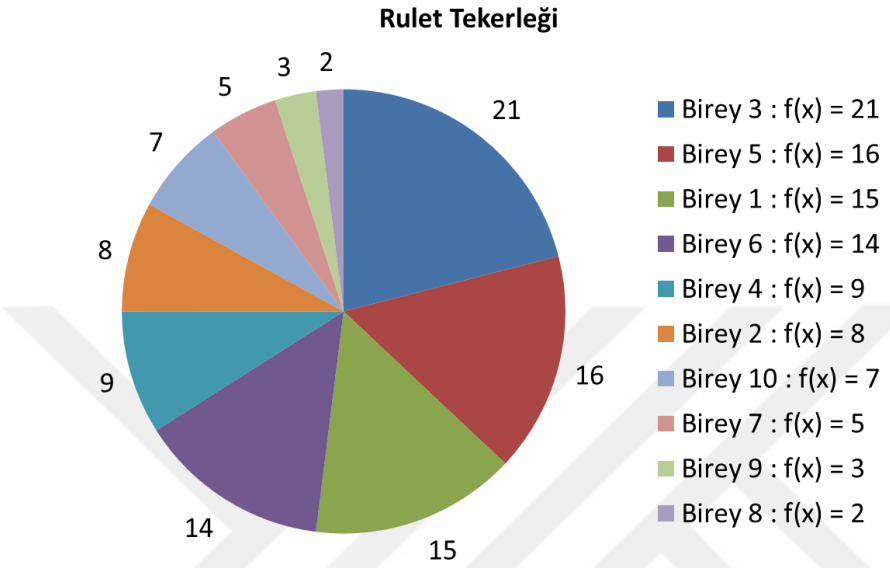
Mevcut nüfus kullanılarak yeni nüfusun oluşturulması için çaprazlama ve mutasyon işlemlerine tabi tutulacak bireylerin belirlenmesi gerekmektedir. GA'nın temel prensibine göre iyi bireyler ebeveyn olarak kullanılarak yeni bireyler oluşturulmalıdır. Kötü bireylerin elenmesi, iyi bireylerin oluşması ve yeni nesillere aktarılması amacıyla iyi bireyleri belirlemek için bir seçim yapılması gerekmektedir. Seçim işlemi 3 adımdan oluşmaktadır; Birinci adım tüm bireylerin amaç fonksiyon değerlerinin hesaplanması, ikinci adım bireylere amaç fonksiyonu değerlerine göre uygunluk değerlerinin atanması, üçüncü adım ise bireylerin sahip oldukları uygunluk değerlerine göre seçilmeleri ve yeni birey üretimi için eşleştirme havuzuna atılmalarıdır (Tuncer ve Yıldırım, 2012).

En çok bilinen seçim yöntemleri arasında rulet tekerleği seçimi, turnuva seçimi, sıralı seçim yöntemi yer almaktadır. Tüm seçim yöntemlerinde, uygunluk değeri iyi olan bireylerin seçilme olasılıkları da fazla olmaktadır.

3.5.1 Rulet tekerleği seçim yöntemi

Rulet tekerleği seçimi yöntemi, rastgele bir şekilde rulet tekerleğinin döndürülmesi sonucunda belirlenen noktanın hangi alanın üzerinde duracağı örneğine benzetilebilir (Url-3). Seçim işleminde bireyler için uygulanan *f(x)* amaç fonksiyonunun sonucundaki uygunluk değerlerini kullanır. Şekil 3.3'te örnek bir rulet tekerleği gösterilmektedir. Örnekte *f(x)* değeri en büyük olan bireyin en iyi birey olduğu kabul edilirse, "Birey 3" en iyi birey olmaktadır. *f(x)* değerleri yüksek olan bireylerin uygunluk değerleri de yüksek olacağından, rulet tekerleğinin dilimleri arasında

diğerlerine göre daha büyük paya sahip olmakta ve seçilme olasılıkları da diğer bireylere göre daha yüksek olmaktadır. Bu yöntem düşük uygunluk değeri olan bireye daha az seçilme şansı, yüksek uygunluk değeri olan bireye de daha çok seçilme şansı tanımaktadır. Bundan dolayı yüksek uygunluk değerine sahip bireyin tüm nüfus üzerinde baskın ve egemen olmasına sebep olabilmektedir (Melanie, 1999).



Şekil 3.3 : Rulet tekerleği ile seçim örneği.

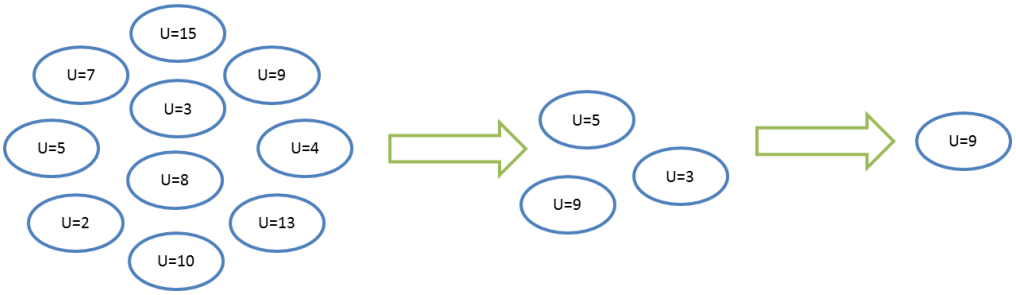
3.5.2 Rank seçim yöntemi

Bu seçim yönteminde bireylere uygunluk değerlerine göre sıralı şekilde değerler verilmektedir. Uygunluk değerlerine göre küçükten büyüğe doğru sıralı olan bir nüfusu düşünülürse, uygunluk değeri en kötü olan bireye 1, sonrakine bireye 2, sonrakine 3 değeri verilerek bu şekilde tüm bireylere sırayla rank değeri verilir. Nüfus sayısının *n* olduğu düşünülürse en iyi birey de *n* rank değerini alır. Bu verilen değerler aynı zamanda bireylerin seçilme şansını doğrudan etkiler. Bireyler tekerlek üzerinde sahip oldukları rank değerlerine göre alan kaplarlar. Bu sayede rulet tekerleği seçim yöntemine göre en iyi birey nüfusa egemen olamayacak ve çözüm uzayındaki arama işleminin genişlemesi rank seçimi yöntemi ile sağlanabilecektir (Er, 2013).

3.5.3 Turnuva seçim yöntemi

Bu yöntemde mevcut nüfustaki bireylerden rastgele belirli sayıda birey seçilir. Bu bireyler kendi aralarında uygunluk değerlerine göre turnuvaya girer ve uygunluğu en yüksek olan birey turnuvayı kazanarak seçilmiş olur. Örneğin rastgele 3 birey seçimi

sonucunda 2. birey, 7. birey ve 9. bireyin turnuvaya gireceği düşünülürse bu bireyler arasında uygunluk değerleri karşılaştırılır ve en iyi olan birey seçilir. Aşağıdaki şekilde uygunluk değeri *U* ile gösterilerek, turnuva seçim yöntemi ile seçilen birey örnek olarak gösterilmektedir. Turnuva seçim örneği Şekil 3.4'de gösterilmektedir.



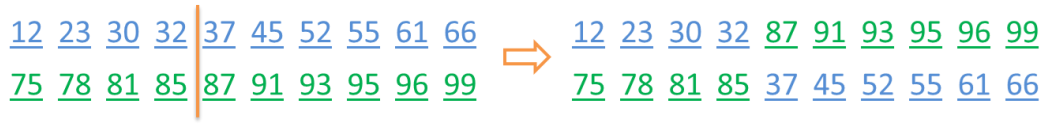
Şekil 3.4 : Turnuva seçim yöntemi örneği.

Seçim yöntemlerinin farklı özellikleri vardır. Farklı özelliklere sahip bu seçim yöntemlerine göre gen havuzundaki çeşitliliğin zamanla azalması söz konusu olabilir. Nüfus içinde bireylerin benzerlikleri fazla olursa bu durum farklı olasılıkların değerlendirilmemesi ve yerel optimum değere takılma gibi riskleri de beraberinde getirebilir. Problemin büyüklüğü ve karmaşıklığına göre uygun seçim yönteminin belirlenmesi gerekmektedir. Aynı probleme göre farklı seçim yöntemleri denendiğinde bu seçim yöntemlerinin de birbirleri arasında avantaj ve dezavantajları olabilmektedir (Razali ve Geraghty, 2011).

3.6 Çaprazlama

Daha iyi bireylerin üretimi için seçilen ebeveynlerdeki bazı genlerin karşılıklı olarak değiştirilmesiyle oluşan işlemdir. Çaprazlamadaki amaç uygunluk değeri daha iyi olan çocuk bireylerin üretilmesidir (Anand ve Spears, 1991). Çaprazlama işlemi GA için büyük bir öneme sahiptir çünkü algoritmanın arama uzayının araştırılmamış kısımlarına erişimini çaprazlama işlemi sağlamaktadır (Gen ve Cheng, 1997).

Çaprazlama işlemi rastgele belirlenen bir çaprazlama noktasına göre yapılır ve bu noktaya göre belirlenen bireylerin genleri karşılıklı takas edilir. Çaprazlama işlemi genel olarak tek noktadan yapılıyor olsa da problemin yapısına göre birden fazla noktadan yapılan çalışmalar da mevcuttur. Şekil 3.5'de örnek olarak verilen tek noktalı bir çaprazlama işleminde ebeveyn bireylerin 4. geninden itibaren çaprazlandığı ve bu noktadan itibaren karşılıklı olarak genlerinin değiştirildiği görülmektedir.



Şekil 3.5 : Çaprazlama örneği.

3.7 Mutasyon

Mutasyon işleminde bireye ait genler belirli bir mutasyon oranına göre rastgele olarak değiştirilir. Mutasyon işlemi, bireyler birbirlerine benzemeye başladığında ve aynı bireyler üzerinde kısır döngü oluşmaya başladığında, GA'yı sıkıştığı bu durumdan kurtarmak için kullanılır. Mutasyon işlemi ile nüfus içindeki birey çeşitliliğinin artması amaçlanmaktadır. (Yang, 1997). Bu açıdan algoritmanın yerel optimum değerlere takılması önlenerek daha geniş bir arama uzayında çözüm aranmaktadır.

Mutasyon işlemi için kullanılan olasılık değeri, gereğinden fazla büyük olursa, mutasyon işlemi bireyin sahip olduğu her bir gen için meydana gelebilir ve bu durum algoritmayı rastsal bir aramaya dönüştüreceği için çözüme ulaşmak da zor bir hale gelebilir. Mutasyon işlemi için kullanılan olasılık değeri az olursa bu durumda arama uzayının farklı noktaları değerlendirilememiş olacaktır. Bu sebeplerden dolayı mutasyon olasılık değerini doğru ayarlamak gerekir. Mutasyon olasılık değeri genellikle <= %1 olacak şekilde belirlenmektedir. Şekil 3.6'da verilen örnek bir mutasyon işleminde verilen mutasyon oranına bağlı olarak bireydeki 8. genin mutasyona uğrayıp, rastgele değiştirildiği görülmektedir.

Şekil 3.6 : Mutasyon örneği.

3.8 Elitizm

Çaprazlama ve mutasyon işlemleriyle oluşan yeni bireylerde en iyi bireyi kaybetme olasılığı bulunmaktadır. Elitizm yöntemi kullanılarak bu durumun önüne geçilir ve en iyi bireyler saklanarak sonraki nesle aktarılır. Elitizm işleminde, her nesilde nüfus içerisindeki bireylerden amaç fonksiyon değeri en kötü olan birey ile bir önceki nesilde saklanan en iyi birey yer değiştirilerek, elit birey yeni nüfusa katılmaktadır. Bu sayede en iyi bireyin kaybolması önlenmiş olmaktadır.

3.9 Amaç Fonksiyonu

GA'da bireyin kalitesini belirlemek için kullanılan en önemli fonksiyondur. Mevcut nüfustaki her birey için amaç fonksiyonuna göre uygunluk değerleri bulunur. Bireyin sahip olduğu bu uygunluk değeri, bireyin kalitesini gösterirken aynı zamanda çözüme ne kadar yakın olduğunu da ifade etmektedir. Amaç fonksiyonu probleme göre farklılık gösterebilen ve hedeflenecek optimum çözümü sağlayabilecek bir fonksiyondur (Tang ve diğ., 1996, Deb ve diğ., 2002).

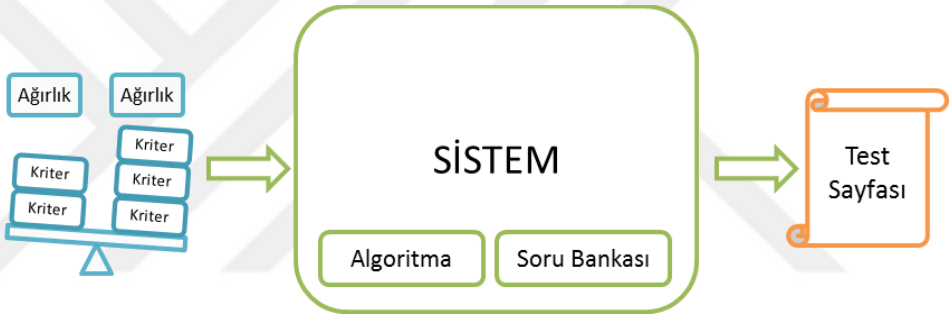
3.10 Algoritma Sonlandırma İşlemi

Algoritmanın çalışacağı süreyi veya algoritmanın iterasyon sayısını sınırlandırmak amacıyla kullanılan ve algoritmanın çalışmasını durduran işlemdir. Sonlandırma kriteri, algoritmada iterasyonlar devam ederken nüfusta çeşitlilik olmadığında, aday çözümlerde bulunan en iyi birey değişmediğinde algoritmanın daha fazla zaman harcanmaması amacıyla uygun şekilde belirlenebilir. Bu kriter maksimum iterasyon sayısı olarak belirlenebileceği gibi beklenen kalite değerine göre de olabilir. Örneğin en iyi çözümde amaç fonksiyon değerinin 0,01 gibi bir değer olması (istenen çözümün 0 olduğu varsayılırsa, 0 değerine çok yaklaşması) çözüm için yeterli kabul edilerek sonlandırma kriteri olarak verilebilir.

4. GENETİK ALGORİTMA İLE TEST SAYFASI OLUŞTURMA

4.1 Genetik Algoritma ile Test Sayfası İlişkilendirmesi

Test sayfası üretimi çoklu kısıtlar arasında istenilen şekilde soruların tespit edilmesini gerektiren çok amaçlı bir optimizasyon problemi olarak ele alınabilir. Yapay zekâ algoritmalarından biri olan GA'nın kullanıldığı bu problem için farklı kriterlere göre (soru sayısı, bölümler, zorluk derecesi, bilgi puanı, cevaplama süresi ve seçilme sıklığı gibi) adaptif şekilde en iyi çözüm aranabilmektedir. Problemin çözümü için oluşturulan sistemin modeli Şekil 4.1'de gösterilmiştir.



Şekil 4.1 : Sistem modeli.

GA'daki terminoloji ile test sayfası karşılaştırıldığında Çizelge 4.1'deki gibi bir ilişki tablosu ortaya çıkarılabilir.

Çizelge 4.1 : GA – Test Sayfası İlişki Tablosu.

GA	Test Sayfası	Açıklama
Nüfus	Test sayfaları Kümesi	Algoritma için başlangıçta belirlenen test sayfası aday sayısı
Kromozom	Test sayfası	Her bir test sayfası adayı
Gen	Soru	Test içindeki bir soru
Gen özellikleri	Soru öznitelikleri	Test içindeki sorunun özellikleri

Test sayfası sorularında her bir soruya ait soru ID, soru metni, sorunun bölümü, zorluk derecesi, soru puanı, bilgi puanı, seçilme sıklığı, çözüm süresi, yetenek seviyesi gibi öznitelikler bulunmaktadır. Bu bilgiler soru bankası veri tabanında bulunmaktadır. Soru bankasında n adet soru olduğu ve her bir sorunun m adet

özniteliği olduğu esas alındığında tüm sorulara ait bilgileri barındıran *S* matrisi aşağıdaki gibi ifade edilebilir (Wu ve Song, 2009).

$$S = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} \quad \textbf{(1)}$$

Tüm sorulara ait bilgilerin olduğu *S* matrisinde her bir satır bir test sorusunu ve o soruya ait olan öznitelikleri ifade etmektedir. İstenilen çoklu kısıtlamalara uymak için test sayfasında yer alacak her bir soruya ait bu özniteliklerin değerlendirilmesi gerekmektedir. Çoklu kısıtlamalarla beraber bu durum ise istenilen test sayfasının hazırlanması için bir optimizasyon problemini ortaya çıkarmaktadır. Bu tez çalışmasında, test sayfası üretiminde kullanılacak soruların en önemli özniteliklerinden olan sorunun bölümü, sorunun zorluk seviyesi, sorunun puanı, sorunun bilgi puanı, sorunun seçilme sıklığı, sorunun cevaplama süresi değerleri ve bu değerlerin test içindeki ağırlık katsayıları esas alınmaktadır. Test sayfası için istenilen çoklu kısıtlamalara bağlı olarak ve sayfanın istenilene ne kadar yakın olduğunu anlamaya yönelik, aşağıdaki formüllerde hata değerleri belirlenmektedir. Bu hesaplanacak olan hata değerleriyle beraber daha sonra ilgili ağırlık katsayıları da kullanılarak test sayfası adayının istenilen çoklu kriterlere ne kadar uyuyor olduğu ve buna bağlı olarak test sayfasının kalitesi belirlenmektedir. Test sayfasının kalitesi belirlenirken, amaç fonksiyonu içinde kullanılacak olan hata değerlerine ait formüller bu tez çalışması kapsamında oluşturulmuş ve aşağıdaki gibi *e* hata değerleriyle ifade edilmiştir.

$$e_1 = \left| Dd - \frac{\sum_{i=1}^{n} a_{i1} a_{i2}}{\sum_{i=1}^{n} a_{i2}} \right| \quad \textbf{(2)}$$

$$e_2 = \left| Dk - \frac{\left(\sum_{i=1}^{n} a_{i3}\right)}{n} \right| \quad \textbf{(3)}$$

$$e_3 = \left| \frac{\left(\sum_{i=1}^{n} a_{i4}\right)}{n} \right| \quad \textbf{(4)}$$

$$e_4 = \left| \frac{Ds - \left(\sum_{i=1}^{n} a_{i5}\right)}{n} \right| \quad \textbf{(5)}$$

Bu formüllerde;

e_1: Sorunun zorluk seviyesi için hesaplanan hata değerini (2),

e_2: Sorunun bilgi puanı için hesaplanan hata değerini (3),

e_3: Soruların seçilme sıklığının en az olması için hesaplanan hata değerini (4),

e_4: Soruların toplam süresi ile istenen süre arasındaki hata değerini (5),

Dd: İstenilen zorluk derecesi faktörünü,

Dk: İstenilen bilgi puanı faktörünü,

Ds: İstenilen toplam süre faktörünü ifade etmektedir.

a_{i1}: Sorunun zorluk derecesini,

a_{i2}: Soru puanını,

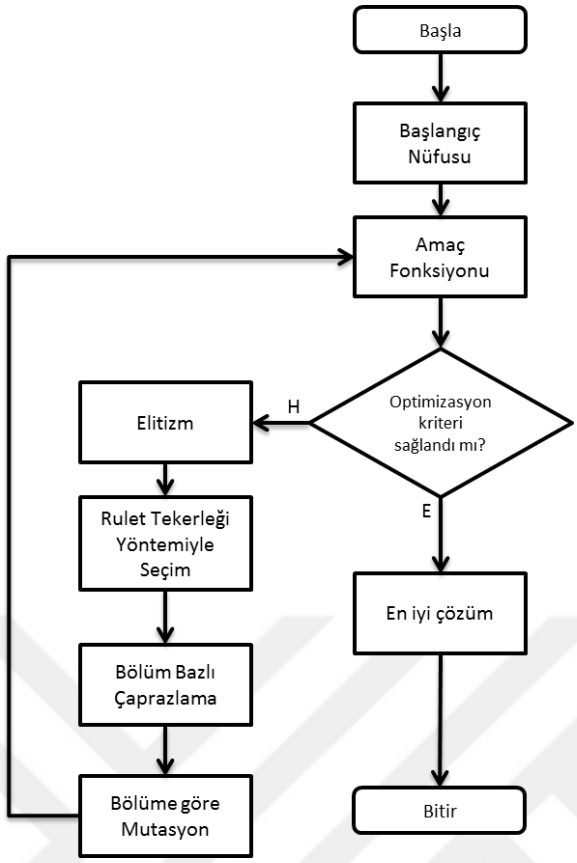
a_{i3}: Sorunun bilgi puanını,

a_{i4}: Sorunun seçilme sıklığını,

a_{i5}: Sorunun cevaplama süresi gibi sorulara ait öznitelikleri ifade etmektedir.

Bilgi puanı; sorunun çözülmesi için ne kadar bilgi gerektirdiği, bölüme ait bilgi seviyesini belirtmektedir. Seçilme sıklığı; sorunun daha önceki test sayfalarında kullanılma oranını belirtmektedir. Test sayfası oluşturma işleminde daha önce sorulmamış olan sorulara öncelik verilmesi dikkate alınması gereken ölçütlerden biridir. Sorunun bölümü; sorunun ait olduğu bölümü göstermektedir. Bu öznitelik soruların bölümlere göre uygun olarak seçilmesini doğrudan etkileyen faktördür.

Şekil 4.2'de tez çalışmasında kullanılan GA ile test sayfası üretimi işlemi için akış diyagramı gösterilmiştir. Akış diyagramına göre; başlangıç nüfusu oluşturulduktan sonra her bir aday çözümün amaç fonksiyon değerleri hesaplanır. Optimizasyon kriteri sağlandığı anda algoritma sonlandırılır ve bulunan en iyi aday çözüm, problemin çözümü olarak saklanır. Optimizasyon kriteri sağlanmadıysa elitizm işlemi uygulanır. Daha sonra bireyler amaç fonksiyon değerlerine bağlı olarak sahip oldukları uygunluk değerlerine göre rulet tekerleği yöntemi ile seçim işlemine tabi tutulurlar. Seçim işleminden sonra seçilen bireylere bölüm bazlı olarak çaprazlama ve mutasyon işlemleri uygulanır. Mutasyondan sonra oluşan yeni nüfusun amaç fonksiyon değerleri hesaplanıp, algoritma bu şekilde durdurma kriteri sağlanana kadar iteratif olarak çalışır.

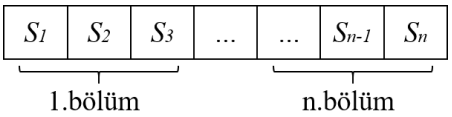


Şekil 4.2 : GA - test sayfası üretimi akış diyagramı.

4.2 Başlangıç Nüfusu Oluşturma

GA'da başlangıç nüfusu genellikle rastgele oluşturulur. Tez çalışmasında, her bir birey (kromozom) bir test sayfasını ve bireydeki genler de test sayfasındaki soruları temsil etmektedir. Soru özelliklerinde bölüm bilgisi de mevcuttur ve her bir bölüme ait soru numaraları da belirlidir. İstenilen bölümlere ait belirli sayıda soru üretimi göz önüne alındığında, başlangıç nüfusu oluşturulurken tamamen rastgele bir nüfusun oluşturulması çözüm için gereken zamanı da arttırmaktadır. Aday çözüm olan test sayfasındaki bölümler için istenilen sayıda soru olması beklenirken, bölümler baz alınmadan rastgele soru seçimi yapılması bölümlere ait soru sayılarının da istenilenden farklı olmasına sebep olmaktadır. Bu durumlar göz önüne alındığında başlangıç nüfusu oluşturulurken tamamı rastgele oluşturulan bir nüfus yerine, seçili bölümlerden istenen soru sayısına göre ilgili bölümler için rastgele nüfus oluşturma gerçekleştirilmiştir. Soruların diğer özelliklerinden olan zorluk seviyesi, puanı, bilgi puanı, seçilme sıklığı gibi özellikler başlangıç nüfusu oluştururken kullanılmamakta olup bu özellikler amaç fonksiyonunda kullanılmıştır.

Nüfus oluşturma işleminde üretilen aday test sayfaları (bireyler) ikili kodlamanın daha fazla yer kaplamasından, hesaplama süresinin artmasından ve program kolaylığından dolayı tam sayı kodlama ile kodlanarak oluşturulmuştur. Şekil 4.3 çalışmada *n* adet sorudan oluşan bir test sayfası oluşturma işleminde kullanılan örnek bir test sayfası için kromozomu göstermektedir.



Şekil 4.3 : Test sayfası için kromozom gösterimi.

4.3 Amaç Fonksiyonu

Test sayfası oluşturma problemi için oluşturulacak test sayfasının kalitesini ve istenilen özelliklerde olmasını belirleyen en önemli metot amaç fonksiyonudur. Test sayfası oluşturma problemindeki asıl amaç; istenilen bölümler içerisinden istenilen sayıda soru seçilmesi ve bu seçilen soruların öznitelikleri ve istenilen çoklu kısıtlar göz önüne alınarak bir test sayfasının optimum şekilde oluşturulmasıdır. Bu tez çalışmasında amaç fonksiyonu için bölüm 4.1'de açıklanan formüllerde belirtilmiş olan her bir hata değeri ile kısıtları kullanıcı tarafından belirlenmiş olan ağırlık katsayıları çarpılır, bu çarpımların sonucu toplanarak problemin amaç fonksiyon değeri hesaplanır. Bu amaç fonksiyonu sonucunda 0'a en yakın olan sonuç en az hata değerine sahip olacağından en iyi sonuç olarak kabul edilmektedir. Amaç fonksiyonu denklem **(6)** ile belirtilmiştir.

$$f = \sum_{i=1}^{4} w_i e_i \qquad \textbf{(6)}$$

Bu formülde;

f: amaç fonksiyon değerini,

e_i: i. kısıt fonksiyonunun hata değerini,

w_i: i. hata değeri için ağırlık katsayısını ifade etmektedir.

Test sayfası oluşturma işleminde istenilen çoklu kriterlerin çözüm kalitesine etki oranı w_i ağırlıkları ile belirlenmektedir. Kullanıcı tarafından belirlenen w_i ağırlıklarının toplamı olan w_t denklem **(7)**'deki gibi ifade edilmiş olup bu değerlerin toplamı 1'e eşit olacak şekilde belirlenmiştir.

$$w_t = \sum_{i=1}^{4} w_i = 1 \quad \textbf{(7)}$$

Bu formülde;

w_1: zorluk seviyesi için ağırlık katsayısını,

w_2: bilgi puanı için ağırlık katsayısını,

w_3: seçilme sıklığı için ağırlık katsayısını,

w_4: cevaplama süresi için ağırlık katsayısını ifade etmektedir.

Amaç fonksiyonunda kullanılan ve isteğe göre belirlenebilen zorluk derecesi, bilgi puanı, cevaplama süresi, seçilme sıklığı kriterleri ve bu kritlerlerin ağırlık çarpan değerleri (w_1, w_2, w_3, w_4) ile kullanıcının istediği özelliklerde bir test sayfası üretilmesine imkan sağlanmaktadır. Test sayfası üretiminden önce zorluk derecesi ve seçilme sıklığı kriterleri önemliyse bu kriterlere ait ağırlık çarpan değerleri yüksek olarak girilir. Bu sayede oluşturulan test sayfasının zorluk derecesi ve geçmişte sorulmuş sorulardan oluşmaması kriterlerine daha çok uyması sağlanır. Benzer şekilde zorluk derecesi ve bilgi puanı kriterleri diğer kriterlerden daha önemliyse bu durumda bu kriterlere ait ağırlık çarpan değerleri yüksek olarak girilir ve test sayfasının bu kriterlerle belirlenen özelliklere daha çok uyması beklenir. Kriterlerden seçilme sıklığı kriterinin bir önemi yoksa ve test sayfasında geçmişte çıkan soruların olması önemli değilse, bu kritere ait ağırlık çarpan değeri 0 olarak girilir, bu durumda algoritma için bu kriter önemsiz ve etkisiz hale getirilmiş olur. Kriterler için kullanılan bu ağırlık çarpanları ile esnek bir yapı oluşturmak hedeflenmiştir.

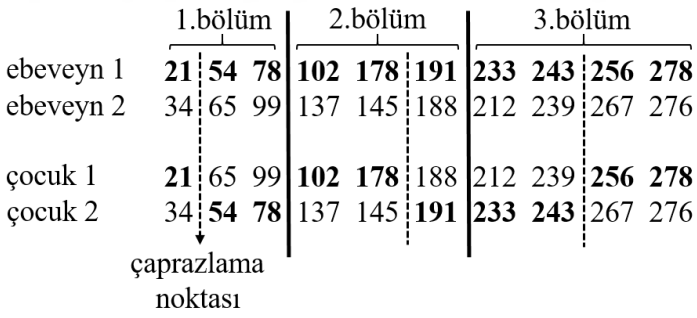
4.4 Seçim

Tez çalışmasında kaliteli bireylere öncelik verme amaçlı yapılacak seçim işlemi için GA'da yaygın olarak kullanılan rulet tekerleği metodu kullanılmıştır. Bireylerin uygunluk değerleri sonuçlarına göre rulet tekerleğinde sahip olacakları dilimler için yüzdeleri hesaplanır. Bu yüzdelik değerlerine göre bireylere büyükten küçüğe doğru sıralama işlemi yapılır. Rastgele üretilmiş 1-100 arasındaki bir sayının hangi bireyin yüzdelik olasılık değerinde olduğuna bakılarak birey seçim işlemi yapılır. Örneğin; en iyi bireyin yüzdelik diliminin %21 sonraki en iyi 2. bireyin %16 olduğunu düşünürsek, 1 ile 100 arasında rastgele seçilmiş olan ve 0-21 arasında gelen her sayı en iyi bireyi, 22-37 arasında gelen her sayı ise en iyi 2. bireyi işaret edecektir.

Bireyler seçim işlemi yapılacağı zaman bu şekilde seçilme olasılık değerlerine göre seçilirler, bu sayede iyi bireylerin seçilme şansı diğerlerine göre daha çok olmakta ve iyi bireylerin genleri kullanılarak nüfusun daha da iyileşmesi amaçlanmaktadır.

4.5 Çaprazlama

Tez çalışmasında standart GA'dan farklı olarak bölüm bazlı çaprazlama işlemi uygulanmıştır. Başlangıç nüfusunda bölümlere ait soru sayıları dikkate alınarak alınarak elde edilmiş olan test sayfasının, çaprazlama işlemlerinden sonra bölümlerdeki soru sayılarının değişmeyecek şekilde kalması gerektiği dikkate alınarak bölüm bazlı çaprazlama ihtiyacı olmuştur. Çaprazlama işleminde ilk olarak çaprazlama noktası belirlenmeden önce bireyin sahip olduğu bölüm soruları kendi aralarında gruplanarak sıralanmış ve her bölüme özgü olmak üzere çaprazlama noktaları ayrı ayrı rastgele olacak şekilde belirlenmiştir. Bu aşamadan sonra iki bireyin çaprazlama işlemleri sadece aynı bölümler arasında gerçekleştirilmektedir. Bu sayede bir bölüme ait olan sorunun diğer bölüme ait olan bir soru ile yer değiştirmesi önlenmiş olmaktadır. Şekil 4.4'te çalışmada kullanılan çaprazlama işlemi için bir örnek gösterilmektedir.



Şekil 4.4 : Test sayfası oluşturma için çaprazlama işlemi.

İki ebeveynin çaprazlanması sonucu oluşturulacak çocuk bireyde birden fazla aynı sorudan oluşmamasının da önüne geçilmesi gerekmektedir. Bunun için soru ebeveynden alınmadan önce yeni bireyde daha önce olup olmadığına bakılarak kontrol edilmektedir. Eğer soru yeni bireyde önceden mevcut olan bir soru ise bu durumda bu soru yerine diğer ebeveynin sorusu alınır. Çaprazlama işleminde nüfusdaki birey sayısı kadar iterasyon yapılarak her iterasyonda bir çocuk birey oluşturulmuş ve bu şekilde yeni nüfusdaki birey sayısının da aynı kalması

sağlanmıştır. Şekil 4.5'de çalışmada kullanılan çaprazlama işlemi ile ilgili akış diyagramı verilmiştir.

Şekil 4.5 : Test sayfası oluşturma için çaprazlama akış diyagramı.

4.6 Mutasyon

Standart GA'daki mutasyon işleminin uygulanması tekrarlı soruların oluşmasına veya mutasyona uğrayacak olan sorunun ait olduğu bölüm yerine başka bir bölümden soru seçilmesine sebep olabilmektedir. Tez çalışmasında, mutasyon işleminde standart GA'dan farklı olarak mutasyona uğrayacak olan soru hangi bölümde yer alıyorsa soru havuzundan o bölüme ait rastgele bir soru seçilmektedir. Şekil 4.6'da çalışmadaki mutasyon işlemi ve Şekil 4.7'de mutasyon işlemine ait akış diyagramı gösterilmiştir.

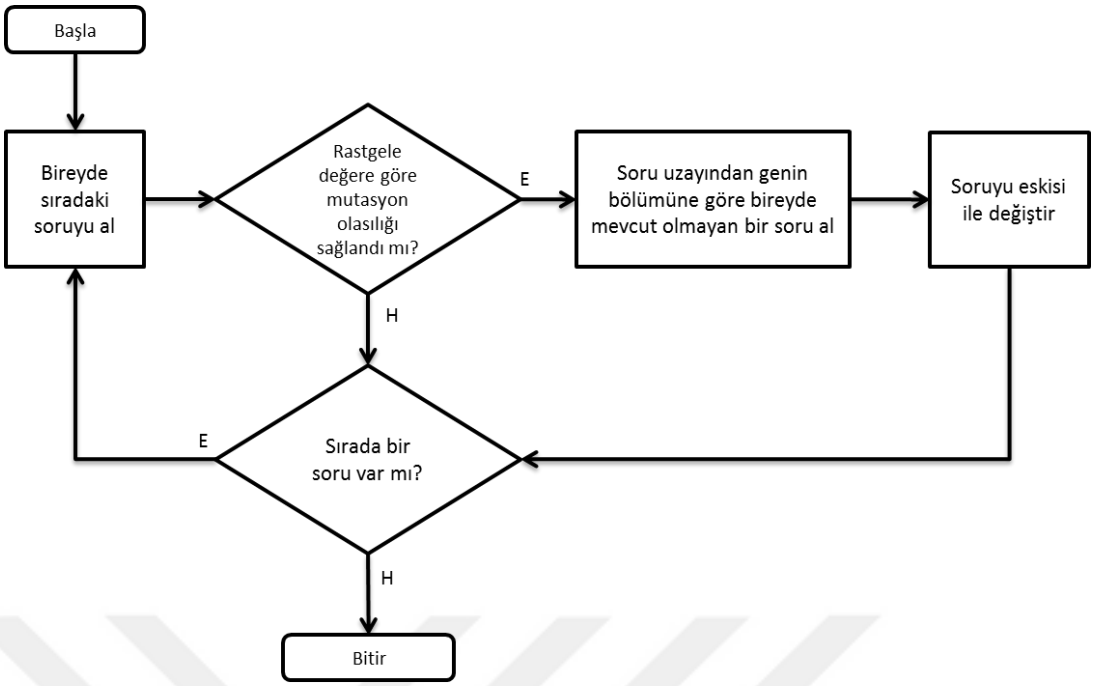
mutasyon öncesi

21 54 78 | 102 178 191 | 233 243 256 278

mutasyon sonrası

21 54 78 | 102 178 **180** | 233 243 **264** 278

Şekil 4.6 : Test sayfası oluşturma için mutasyon işlemi.



Şekil 4.7 : Test sayfası üretimi - mutasyon akış diyagramı.

5. TEST SAYFASI OLUŞTURMA UYGULAMASI

GA ile test üretimi için Web tabanlı bir uygulama gerçekleştirilmiştir. Uygulama için Java, SpringBoot teknolojileri ve Thymeleaf gibi template yapılar kullanılmıştır. Veri tabanı olarak MongoDB (noSql) veri tabanı kullanılmıştır. Uygulama, kullanıcının test sayfası için girebileceği parametre ve kriter arayüzlerine sahiptir. Kullanıcı istediği özellikteki test sayfası için bu ayarları girerek algoritmanın çalışması sonucunda test sayfası oluşturabilmekte ve sonuç olarak üretilen test sayfasındaki soruları görebilmektedir. Uygulama arayüzünde GA ile ilgili ayarlar da belirlenebilmektedir. Soru bankasındaki soruların da görülebileceği arayüzler Türkçe ve İngilizce dillerini destekleyecek şekilde hazırlanmıştır.

5.1 Nesne Tabanlı Model

Test sayfası oluşturma işleminde nesne tabanlı uygulamada kullanılan sınıf yapıları aşağıdaki gibidir;

1- Soru (Gen): id, soru no, zorluk derecesi, puan, tahmini cevaplama süresi, seçilme oranı, bilgi seviyesi puanı, bölüm, soru metni, cevap gibi bilgileri içerir.

2- Test Sayfası (Kromozom/Birey): soru sayısı, ortalama zorluk derecesi, toplam puanı, toplam cevaplama süresi, ortalama bilgi seviyesi gibi bilgileri içerir.

3- Nüfus (Popülasyon): test sayfaları, en iyi test sayfası, istenilen özellikler gibi bilgileri içerir.

4- Özellikler: istenilen soru sayısı, istenilen zorluk seviyesi, istenilen bilgi puanı seviyesi, istenilen bölümlere ait soru sayıları, ağırlık oranları, test sayısı, maksimum iterasyon sayısı, soru bankası soru sayısı, uygunluk fonksiyonu maksimum değeri, son seçilen test soruları gibi bilgileri içerir.

5- Algoritma: çaprazlama oranı, mutasyon oranı, elitizm gibi bilgileri içerir.

5.2 Soru Bankası ve Özellikleri

Kullanılan soru bankasında toplam 2455 soru bulunmaktadır (Url-1). Sorular 5 farklı bölümden oluşur. Her sorunun kendine ait zorluk seviyesi, puanı, bilgi puanı seviyesi, çözüm süresi, geçmişteki sorulma sıklığı gibi özellikler bulunmaktadır. Çalışmadaki soru bankası ve sorular MongoDB veri tabanı üzerinde tutulmuştur. Deneysel çalışmalardaki sonuçlar değerlendirilirken soru bankasının ve sahip olduğu soruların özellikleri önemli olmaktadır. Örneğin çok kolay soruların sayısının çok az olduğu bir soru bankasından, zorluk seviyesi çok kolay bir test elde edilmek istenirse, bu durumda sonuçlar beklenen hata seviyesinin üzerinde olabilir. Kullanılan soru bankasının genel olarak sahip olduğu bilgiler aşağıdaki çizelgelerde gösterilmiştir. Çizelge 5.1'de kullanılan soru bankasının zorluk seviyesine göre dağılımı gösterilmektedir. Soru bankasındaki soruların zorluk seviyeleri 0 ile 4 arasında değişmekte olup zorluk seviyelerinin ortalaması 1,986'dır.

Çizelge 5.1 : Zorluk seviyesine göre soru dağılımı.

Zorluk Seviyesi	Soru Sayısı
Zorluk Seviyesi 0	10
Zorluk Seviyesi 1	937
Zorluk Seviyesi 2	779
Zorluk Seviyesi 3	536
Zorluk Seviyesi 4	193

Soru bankasındaki soruların bilgi puanı seviyelerine göre dağılımı Çizelge 5.2'de gösterilmiştir.

Çizelge 5.2 : Bilgi puanı seviyesine göre soru dağılımı.

Bilgi Puanı	Soru Sayısı
Bilgi Puanı 0	963
Bilgi Puanı 1	773
Bilgi Puanı 2	719

Soru bankasındaki soruların bölümlere göre dağılımı Çizelge 5.3'de gösterilmiştir.

Çizelge 5.3 : Bölümlere göre soru dağılımı.

Bölüm	Soru Sayısı
Bölüm 1	513
Bölüm 2	564
Bölüm 3	551
Bölüm 4	453
Bölüm 5	373

Soru bankasındaki soruların çözüm sürelerine göre dağılımı Çizelge 5.4'te gösterilmiştir.

Çizelge 5.4 : Çözüm süresine göre soru dağılımı.

Çözüm Süresi	Soru Sayısı
1 dk	963
2 dk	773
3 dk	719

Soru bankasındaki soruların seçilme sıklığı değerine göre dağılımı Çizelge 5.5'de gösterilmiştir.

Çizelge 5.5 : Seçilme sıklığına göre soru dağılımı.

Seçilme Sıklığı	Soru Sayısı
0	519
1	491
2	517
3	464
4	464

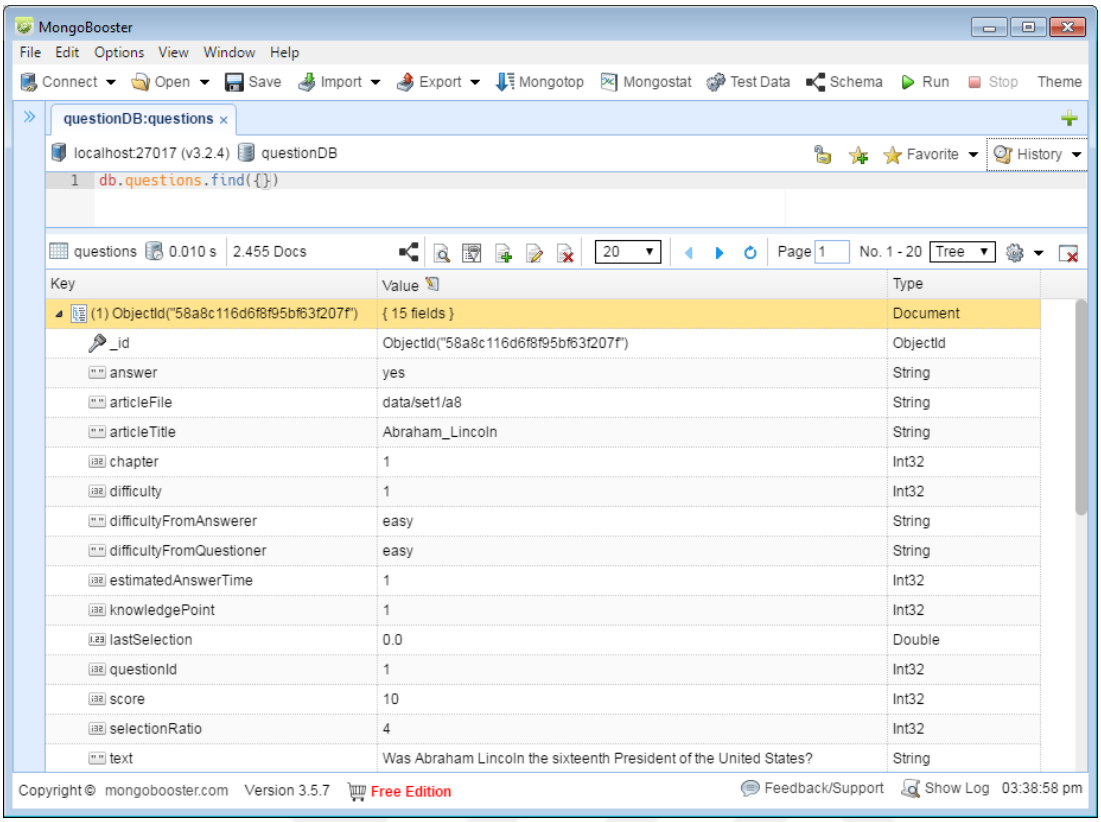
Soru bankasındaki soruların kriterlere göre ortalamaları Çizelge 5.6'da gösterilmiştir.

Çizelge 5.6 : Kriterlerin ortalama değerleri.

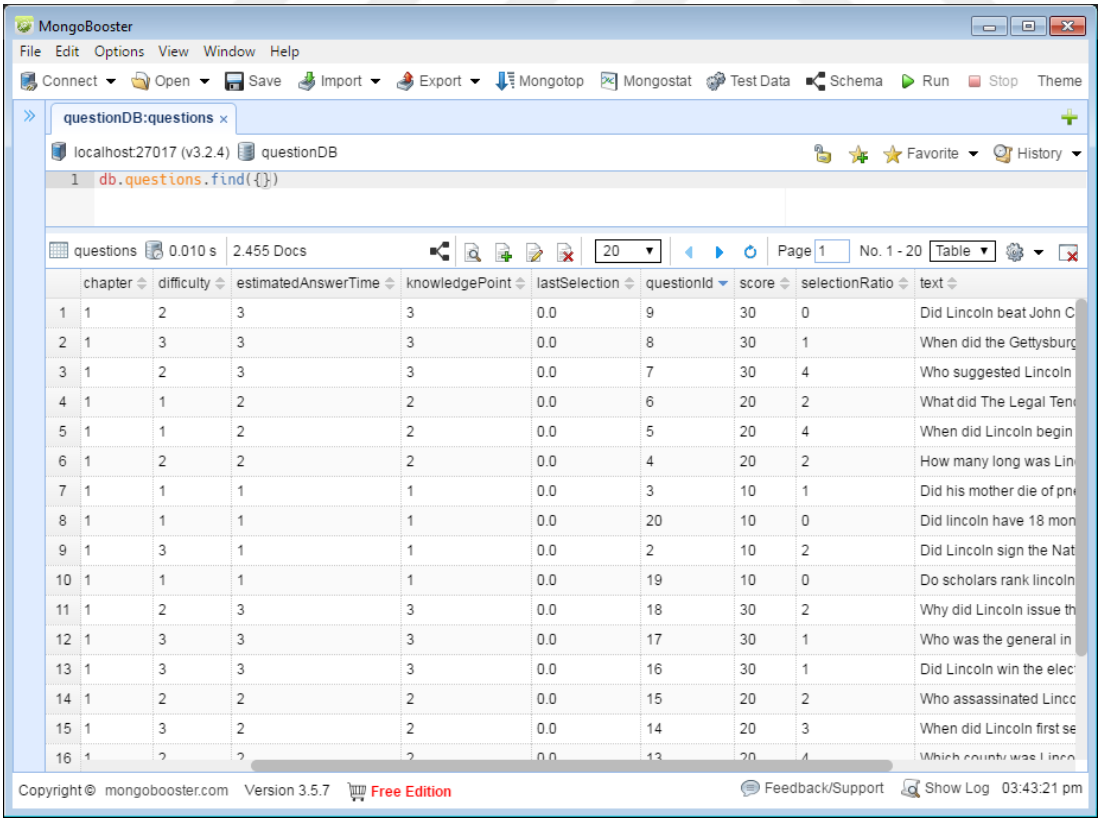
Kriter	Ortalama
Zorluk Seviyesi	1,986
Bilgi Puanı Seviyesi	1,900
Çözüm Süresi	1,900
Seçilme Sıklığı	1,991

Seçilme sıklığı kullanılmasıyla daha önce seçilmemiş sorulara avantaj sağlanarak formüldeki ağırlık oranına göre onlara öncelik verilebilmektedir. Soruların geçmişte sorulma sıklığı bilgisi istenirse tüm sorular için sıfırlanabilmekte ve tüm sorular için arayüzden rastgele olarak da belirlenebilmektedir. Bu tez çalışmasında ise seçilme sıklığı tüm sorular için rastgele belirlenmiş olup deneysel çalışmalarda yanıltıcı sonuçlar oluşturmaması için sabit tutulmuştur.

Veri tabanı olarak kullanılan MongoDB'de arayüz için MongoBooster aracı kullanılmıştır. Bu araçta tek soruya ait özellikler Şekil 5.1'de, tüm sorulara ait özellikler Şekil 5.2'de olduğu gibi görülebilmektedir.



Şekil 5.1 : Bir soruya ait özellikler.

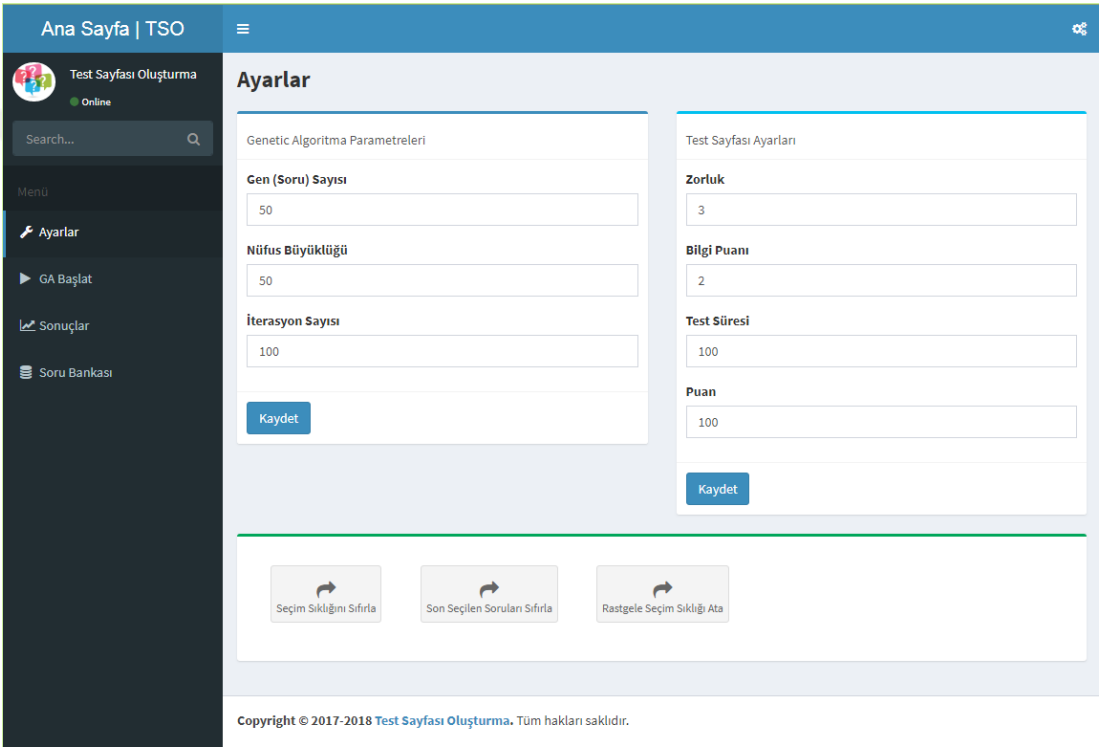


Şekil 5.2 : Sorulara ait özellikler

5.3 Web Uygulaması

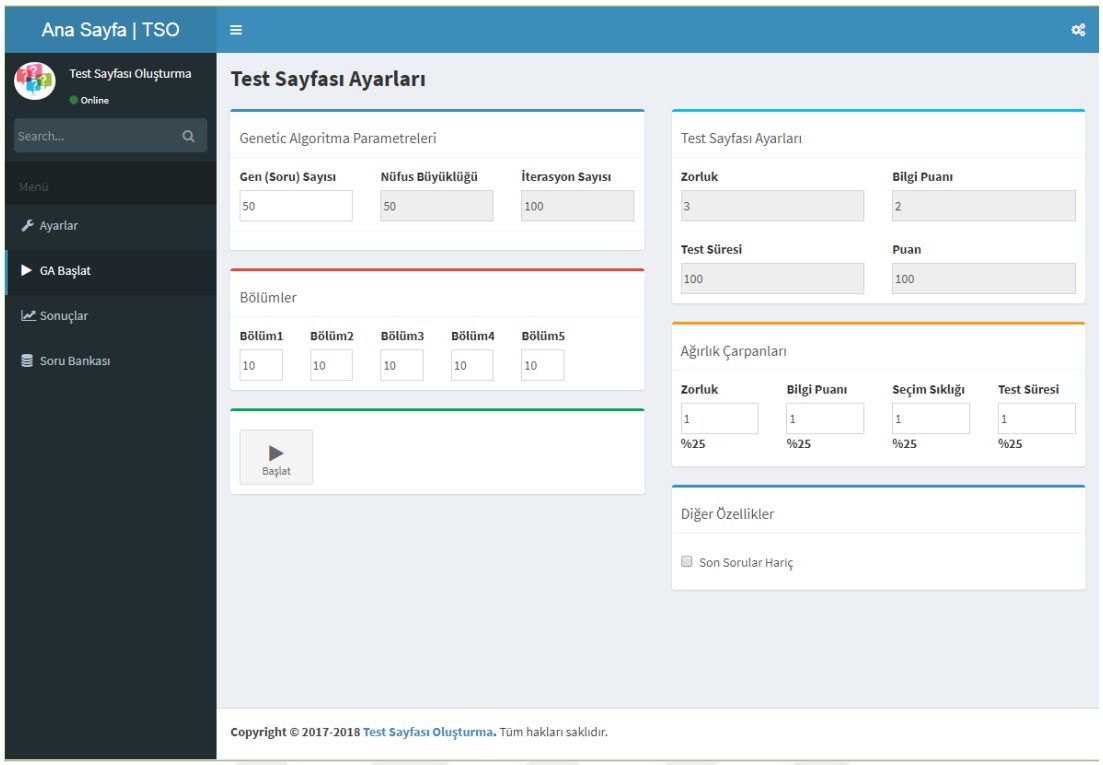
Tez çalışması için gerçekleştirilmiş olan web uygulamasının ayarlar arayüzünde kullanıcı, GA'ya ait gen sayısı, nüfus büyüklüğü, algoritmanın iterasyon sayısı gibi bazı parametreleri girebilmektedir. Ayrıca soruların seçilme sıklığı, son seçilen soru bilgilerini sıfırlayabilmekte ve sorulara rastgele seçim sıklığı atayabilmektedir.

Kullanıcı test sayfası ayarlarıyla ilgili olarak zorluk seviyesi, bilgi puanı, test süresi gibi kriterleri belirleyebilmektedir. Bu kriterlerin belirlenebildiği ayarlar ekranı Şekil 5.3'de gösterilmiştir.



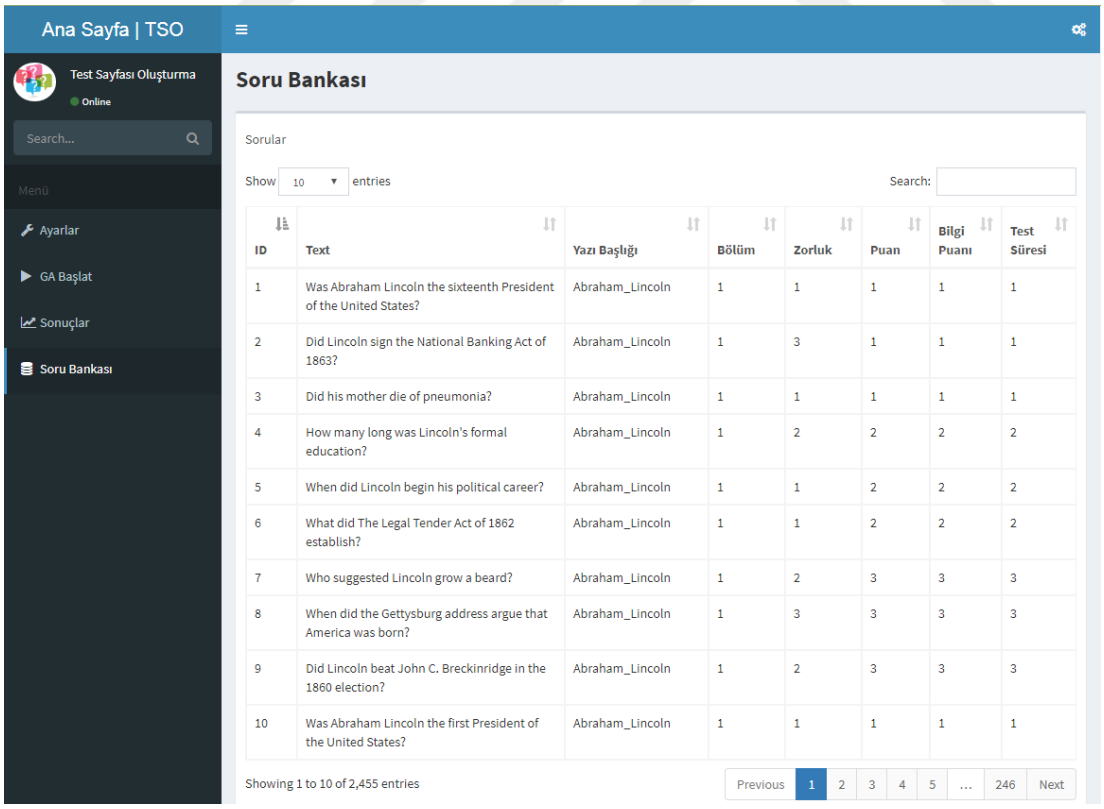
Şekil 5.3 : Ayarlar ekranı.

Tez çalışması için gerçekleştirilmiş olan web uygulamasının "GA Başlat" arayüzünde kullanıcı, test sayfası için istediği soru sayısını girerek bu soruların hangi bölümlere ait olacağını belirleyebilmektedir. Aynı arayüzden test sayfası için zorluk seviyesi, bilgi puanı, seçim sıklığı ve test süresi gibi kriterlerin ağırlık katsayılarını girebilmektedir. "Son Sorular Hariç" özelliği seçildiğinde ise en son sorulmuş olan test sayfasındaki sorular hariç tutularak o sorular olmadan bir test sayfası hazırlanması mümkün olabilmektedir. Ağırlık çarpanı değerlerinin, bölümlere ait soruların belirlenebildiği ekran Şekil 5.4'de gösterilmektedir.



Şekil 5.4 : Test sayfası oluşturma ekranı.

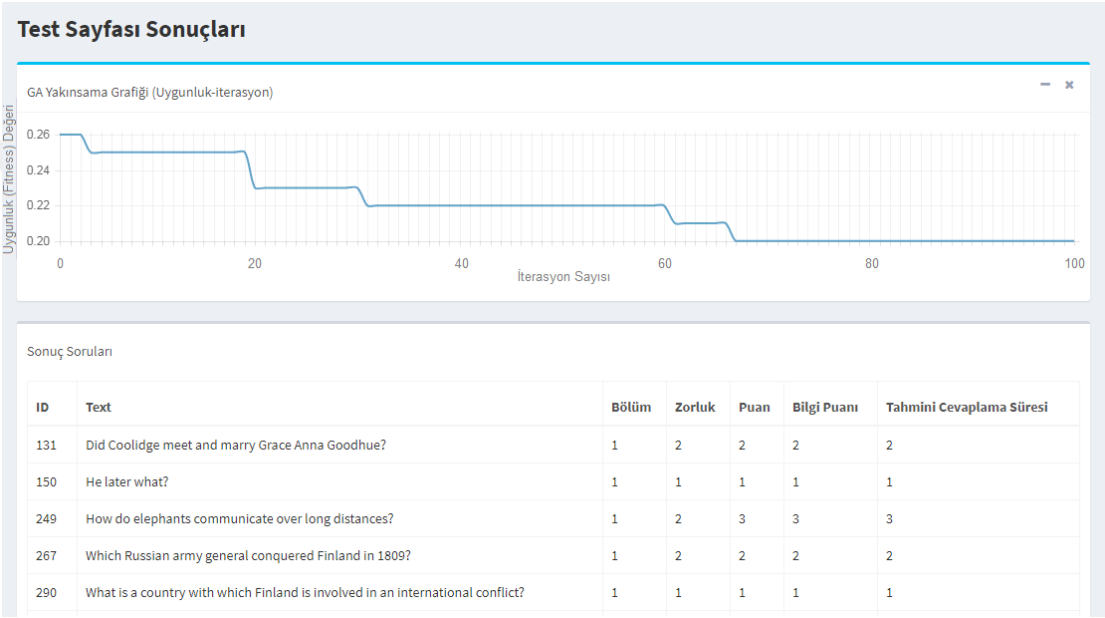
Soru bankasındaki soruların görülebildiği ekran Şekil 5.5'de gösterilmektedir.



Şekil 5.5 : Soru bankası ekranı.

Web uygulamasının “Soru Bankası” arayüzünde soru bankasında yer alan tüm sorular ve onlara ait özellikler görüntülenebilmektedir. Bu sorular özelliğe göre sıralanabilmekte ve sorular içinde arama yapılabilmektedir.

Web uygulamasının “Sonuçlar” arayüzünde ise algoritmanın sonucunda üretilmiş olan test sayfasına ait belirlenmiş sorular görüntülenmektedir. Ayrıca algoritmanın hangi iterasyonunda istenilen sonuca ne kadar yaklaştığı yakınsama grafiği şeklinde gösterilmektedir. Test sonucunun ve testteki soruların yakınsama grafiği ile beraber sunumu Şekil 5.6’da göserilmektedir.



Şekil 5.6 : Test sayfası sonuç ekranı.

5.4 Deneysel Çalışmalar

Deneysel çalışmalarda test sayfalarındaki soru sayıları 20, 50, 100 ve 200 olacak şekilde belirlenmiş ve standart GA ile çalışmadaki GA’nın ürettiği sonuçlar üzerinde karşılaştırmalar yapılmıştır. Sonuçları karşılaştırılan her deneysel çalışma için algoritmalardan her biri 10’ar kez çalıştırılmış ve sonuçların ortalama değerleri çizelgelerde gösterilmiştir.

Nüfus sayısı, iterasyon sayısı, çaprazlama ve mutasyon oranları her iki algoritma için de aynı olacak şekilde belirlenmiştir. Zorluk seviyelerinin sonuçları karşılaştırılırken bilgi puanı seviyesi, seçilme sıklığı ve cevaplama süresi gibi diğer kriterler ve bu kriterlere bağlı ağırlık çarpan değerleri de sabit tutulmuştur. Algoritmaların uygunluk fonksiyonları aynıdır. Böylece aynı soru sayıları ve aynı zorluk seviyelerine sahip olan test sayfalarını üretmek için algoritmaların sonuçları ve performansları incelenebilmiştir. İki algoritma için de nüfus sayısı 80, iterasyon sayısı 100, çaprazlama oranı 1 ve mutasyon oranı ise 0,015 olacak şekilde belirlenmiştir. (Tül ve Tuncer, 2017)

Kriterler için *w* ağırlık çarpanları 0 olarak belirlendiğinde o kriterler dikkate alınmamış olmaktadır. Kritere ait ağırlık çarpanı (yüzdesi) ne kadar büyük değere sahip olursa kriter test için o kadar önemli olmaktadır.

Zorluk seviyesi için algoritmaların karşılaştırılması ve sonuç değerleri Çizelge 5.7'de gösterilmektedir.

Çizelge 5.7 : Zorluk seviyesi için algoritmaların performans karşılaştırması.

Zorluk Seviyesi	Test Soru Sayısı	Nüfus Sayısı	Standart GA Uygunluk Değeri	Çalışmadaki GA Uygunluk Değeri
3	200	80	0,226	0,132
3	100	80	0,203	0,051
3	50	80	0,151	0,046
3	20	80	0,086	0,051
2	200	80	0,057	0,045
2	100	80	0,051	0,038
2	50	80	0,048	0,038
2	20	80	0,059	0,037
1	200	80	0,466	0,248
1	100	80	0,250	0,128
1	50	80	0,206	0,066
1	20	80	0,168	0,055

Çizelge 5.8'de bilgi puanı seviyesi ile ilgili standart GA ve çalışmadaki GA'nın karşılaştırılması verilmiştir. Çizelgeyi oluşturmak için yapılan bu testlerde zorluk seviyesi=2 ve soru ortalama cevaplama süresi=2 değerleriyle sabit tutularak istenilen bilgi puanı kriterinin değiştirilmesi sonucu uygunluk değeri sonuçlarının nasıl değiştiği karşılaştırmalı olarak gösterilmiştir.

Çizelge 5.8 : Bilgi puanı seviyesi için algoritmaların performans karşılaştırması.

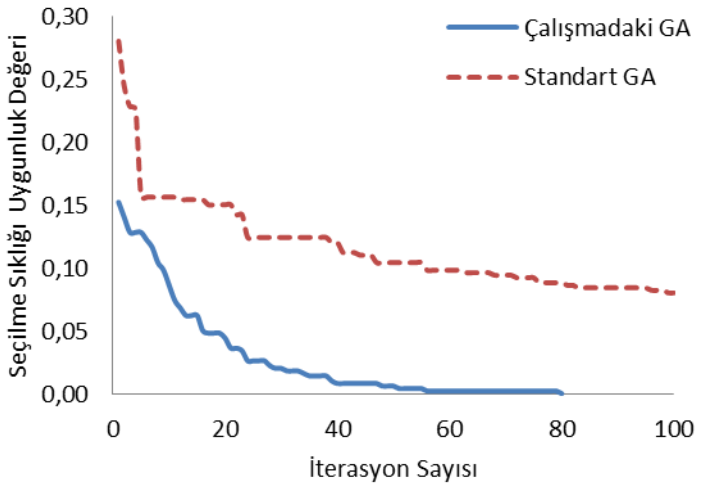
Bilgi Puanı Seviyesi	Test Soru Sayısı	Nüfus Sayısı	Standart GA Uygunluk Değeri	Çalışmadaki GA Uygunluk Değeri
3	200	80	0,303	0,281
3	100	80	0,292	0,271
3	50	80	0,283	0,258
3	20	80	0,252	0,227
2	200	80	0,055	0,048
2	100	80	0,057	0,037
2	50	80	0,053	0,034
2	20	80	0,059	0,036
1	200	80	0,306	0,286
1	100	80	0,303	0,280
1	50	80	0,304	0,275
1	20	80	0,310	0,276

Çizelge 5.9'da seçilme sıklığının ile ilgili standart GA ve çalışmadaki GA'nın karşılaştırılması verilmiştir. Çizelgeyi oluşturmak için yapılan bu testlerde zorluk değeri=2, bilgi puanı değeri=2, soru ortalama cevaplama süresi=2 kriterleri sabit tutularak ağırlık çarpanlarının değiştirilmesi sonucu seçilme sıklığı kritlerinin nasıl değiştiği karşılaştırmalı olarak gösterilmiştir. Çizelgede gösterilmekte olan değerler elde edilirken her bir satır için ortalama değerler alınmıştır. Seçilme sıklığı toplamı, testte yer alan soruların seçilme sıklığı değerlerinin toplamı ile elde edilmiştir. Soru bankasındaki bir sorunun seçilme sıklığı değeri, 0-4 arasındadır ve soru bankasındaki soruların seçilme sıklığı ortalaması ise 1,9'dur. Seçilme sıklığı ağırlık çarpanı (w_3=4) %100 olarak belirlenmiş olan ve farklı sayılardaki testler için çizelgede, çalışmadaki GA kullanılarak soru başına seçilme sıklığı ortalamasının 0'a yakın olduğu görülmektedir.

Çizelge 5.9 : Seçilme sıklığı için algoritmaların performans karşılaştırması.

Seçilme Sıklığı Ağırlık %	Ağırlık Çarpanları				Test Soru Sayısı	Standart GA Sorunun Seçilme Sıklığı Ortalaması	Çalışmadaki GA Sorunun Seçilme Sıklığı Ortalaması
	w1	w2	w3	w4			
%100	0	0	4	0	100	1,202	0,146
%100	0	0	4	0	50	0,728	0
%100	0	0	4	0	20	0,06	0
%50	2	0	2	0	100	1,636	0,714
%50	2	0	2	0	50	1,284	0,348
%50	2	0	2	0	20	0,54	0
%25	3	0	1	0	100	1,898	1,108
%25	3	0	1	0	50	1,436	0,636
%25	3	0	1	0	20	0,84	0,06

Şekil 5.7'de seçilme sıklığı için (w_3) çarpan değeri %100 olarak belirlenerek diğer kriterleri dikkate alınmamış, 50 sorudan oluşması istenilen bir testin yakınsama grafiği görülmektedir. 80. iterasyonda çalışmadaki GA'nın belirlenen tüm kriterlere göre sıfır hata ile çözümü bulduğu görülmektedir.



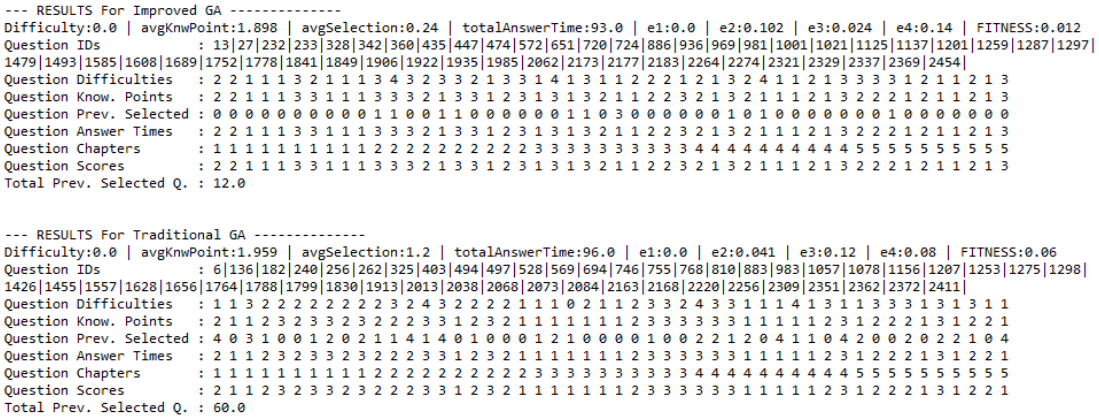
Şekil 5.7 : Seçilme sıklığı için yakınsama grafiği karşılaştırma örneği.

Şekil 5.8'de karşılaştırması yapılmış standart GA ve çalışmadaki GA sonuçlarının olduğu örnek bir konsol ekranı gösterilmektedir. Bu test örneğinde, seçilme sıklığı için kullanılan w_3 ağırlık çarpan değeri %100 olarak belirlenmiş ve 50 soruluk bir test sayfası oluşturulmak istenmiştir. Seçilme sıklığı ile ilgili ağırlık çarpanı w_3 %100 olarak belirlendiği için w_1, w_2, w_4 değerleri 0 olmakta ve zorluk derecesi, bilgi puanı ve cevaplama süresi kriterleri etkisiz hale gelmektedir. Bu ekranda, standart GA ve çalışmadaki GA karşılaştırması amacıyla üretilen test sayfasına ait e_1, e_2, e_3, e_4 hata değerleri, test uygunluk değeri ve sorularla ilgili soru ID, zorluk derecesi, bilgi puanı, soru puanı, seçilme sıklığı, bölümler, cevaplama süreleri ve seçilme sıklığı toplamı gibi bilgiler de detaylı olarak görünmektedir. Sonuca göre standart GA, toplamda 36 defa önceden seçilmiş soru içerirken (ürettiği testteki her bir sorunun daha önce seçilme ortalaması 0,72 iken) çalışmadaki GA hiçbir seçilmiş soru içermemektedir. e_1, e_2, ve e_4 hata değerlerinin ağırlıkları w_1, w_2, w_4 = 0 ağırlık çarpan değerlerine bağlı olarak dikkate alınmamaktadır.

```
--- RESULTS For Improved GA --------------
Difficulty:0.18 | avgKnwPoint:2.143 | avgSelection:0.0 | totalAnswerTime:105.0 | e1:0.18 | e2:0.143 | e3:0.0 | e4:0.1 | FITNESS:0.0
Question IDs            : 35|41|72|170|204|299|336|348|405|486|522|530|567|627|735|773|850|918|947|951|1143|1147|1156|1187|1261|1321|
1360|1420|1443|1608|1642|1720|1733|1786|1808|1822|1845|1927|2005|2012|2087|2104|2307|2329|2346|2350|2380|2420|2440|2443|
Question Difficulties   : 2 1 3 1 2 1 2 2 2 2 4 3 1 1 1 2 3 3 1 1 4 4 1 3 3 1 3 3 2 1 4 1 4 3 1 3 3 2 1 2 3 4 2 1 3 3 3 1 1 1
Question Know. Points   : 2 1 3 1 2 1 3 2 3 3 3 3 1 1 1 3 3 3 1 1 3 3 1 3 3 1 3 3 2 3 3 1 3 3 1 3 3 2 1 1 2 3 2 1 2 3 2 1 1 1
Question Prev. Selected : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Question Answer Times   : 2 1 3 1 2 1 3 2 3 3 3 3 1 1 1 3 3 3 1 1 3 3 1 3 3 1 3 3 2 3 3 1 3 3 1 3 3 2 1 1 2 3 2 1 2 3 2 1 1 1
Question Chapters       : 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5
Question Scores         : 2 1 3 1 2 1 3 2 3 3 3 3 1 1 1 3 3 3 1 1 3 3 1 3 3 1 3 3 2 3 3 1 3 3 1 3 3 2 1 1 2 3 2 1 2 3 2 1 1 1
Total Prev. Selected Q. : 0.0


--- RESULTS For Traditional GA --------------
Difficulty:0.1 | avgKnwPoint:1.878 | avgSelection:0.72 | totalAnswerTime:92.0 | e1:0.1 | e2:0.122 | e3:0.072 | e4:0.16 | FITNESS:0.072
Question IDs            : 54|79|84|136|176|194|249|300|492|507|530|545|592|596|646|832|864|866|959|1038|1126|1190|1192|1248|1259|1369|
1401|1470|1543|1592|1661|1670|1691|1703|1749|1762|1799|1845|1888|2072|2094|2112|2116|2136|2342|2379|2392|2407|2413|2420|
Question Difficulties   : 3 2 0 1 1 3 2 2 2 1 3 1 3 4 4 3 2 3 2 1 2 1 1 2 3 1 3 1 1 1 1 1 1 1 4 1 3 1 3 2 3 1 1 4 1 1 1 2 3 1 1
Question Know. Points   : 3 2 1 1 2 3 3 2 2 1 3 1 3 3 3 3 2 3 3 1 2 1 1 2 3 1 1 1 1 1 1 1 1 1 3 1 3 1 3 1 3 1 1 2 1 1 1 3 3 1 1
Question Prev. Selected : 0 0 1 2 0 1 1 0 0 0 0 1 1 0 1 3 0 0 2 0 0 1 0 0 1 0 2 1 0 0 0 3 1 1 0 3 0 0 1 1 1 0 0 0 2 2 0 3 0 0
Question Answer Times   : 3 2 1 1 2 3 3 2 2 1 3 1 3 3 3 3 2 3 3 1 2 1 1 2 3 1 1 1 1 1 1 1 1 1 3 1 3 1 3 1 3 1 1 2 1 1 1 3 3 1 1
Question Chapters       : 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5
Question Scores         : 3 2 1 1 2 3 3 2 2 1 3 1 3 3 3 3 2 3 3 1 2 1 1 2 3 1 1 1 1 1 1 1 1 1 3 1 3 1 3 1 3 1 1 2 1 1 1 3 3 1 1
Total Prev. Selected Q. : 36.0
```

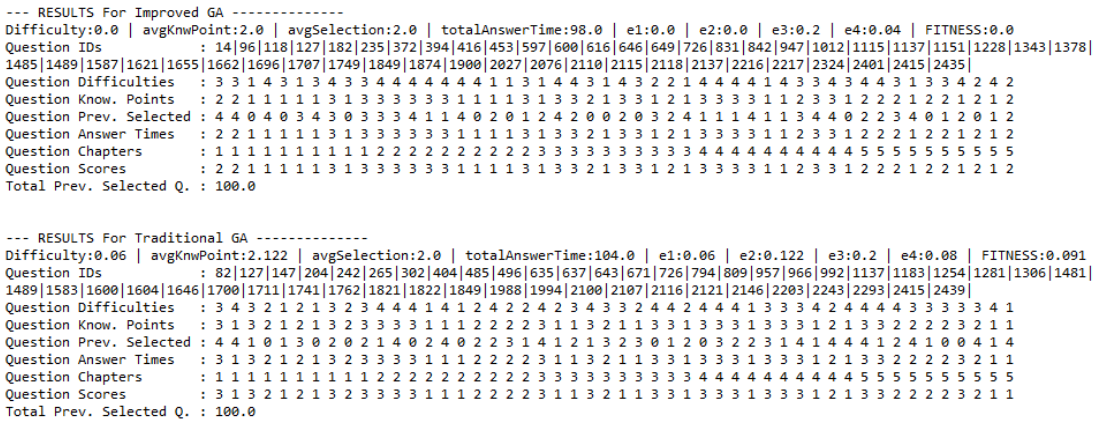
Şekil 5.8 : Seçilme sıklığı ağırlık çarpanı %100 olan sonuç örneği.

Şekil 5.9'da zorluk derecesi ve seçilme sıklığı kriterleri kullanılarak ağırlık çarpan değerleri w_1 ve w_3 %50'şer olarak ayarlanmıştır. Bu örnekte w_2 ve w_4 değerleri 0 yani etkisiz olarak belirlenmiştir. 50 soruluk olan bu örnekte, zorluk seviyesi ve seçilme sıklığı kriterleri yarı yarıya önemli olarak oluşturulan sonuç test sayfasının özelliklerine bakıldığında çalışmadaki GA'nın uygunluk değeri olarak daha başarılı sonuçlar verdiği görülmektedir. Zorluk derecesi olarak iki algoritma sonucu da e_1 hata değerini 0 olarak elde ederek bu kriter için istenileni vermiş olsa da, seçilme sıklığı hata değeri olan e_3 hata değerine bakıldığında çalışmadaki GA'nın daha iyi bir sonuç bulduğu görülmektedir.



Şekil 5.9 : Zorluk ve seçilme sıklığı için ağırlık çarpanı %50 olan sonuç örneği.

Şekil 5.10'da sonuçları verilen karşılaştırma örneğinde, zorluk derecesi ve bilgi puanı kriterleri kullanılarak ağırlık çarpan değerleri w_1 ve w_2, %50'şer olarak ayarlanmıştır. Bu örnekte seçilme sıklığı ağırlık çarpanı w_3 ve cevaplama süresi ağırlık çarpanı w_4 değerleri 0 yani etkisiz olarak belirlenmiştir. Buna göre soruların önceden seçilmiş olması ve testin toplam cevaplama süresi etkisiz olmaktadır. 50 soruluk olan bu örnekte zorluk seviyesi kriteri 3, bilgi puanı kriteri ise 2 olarak girilmiştir. Zorluk seviyesi ve bilgi puanı kriterleri yarı yarıya önemli olarak oluşturulan sonuç test sayfasının özelliklerine bakıldığında GA'nın uygunluk değeri olarak daha başarılı sonuçlar verdiği görülmektedir. Çalışmadaki GA, zorluk derecesi için e_1 hata değerini ve bilgi puanı için e_2 hata değerini 0 olarak bulurken, standart GA e_1 ve e_2 hata değerlerini daha yüksek bulmuştur. Bu durum, iki algoritmanın çalıştırılması sonucunda oluşturdukları test sayfalarının arasındaki farkı göstermektedir.



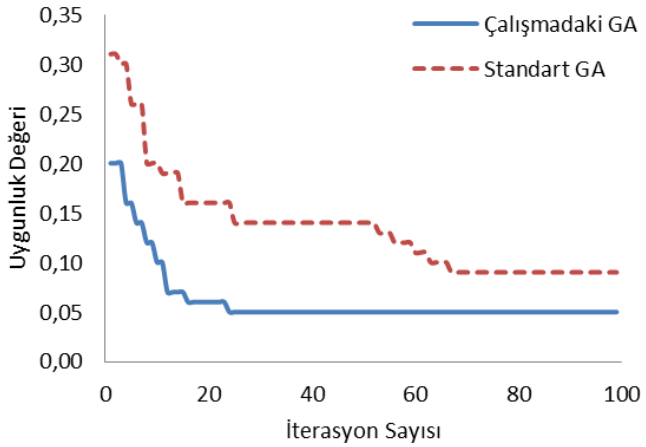
Şekil 5.10 : Zorluk ve bilgi puanı için ağırlık çarpanı %50 olan sonuç örneği.

Şekil 5.11'de sonuçları verilen karşılaştırma örneğinde, zorluk derecesi, bilgi puanı, seçilme sıklığı ve cevaplama süresi kriterleri kullanılarak ağırlık çarpan değerleri w_1, w_2, w_3 ve w_4 hepsi eşit olacak şekilde (her biri %25) ayarlanmıştır. 50 soruluk olan bu örnekte zorluk seviyesi kriteri 3, bilgi puanı kriteri ise 2 olarak girilmiştir. Bu kriterlere göre sonuçlar karşılaştırıldığında çalışmadaki GA'nın daha başarılı olduğu görülmektedir. Çalışmadaki GA'nın zorluk derecesi için kullanılan e_1 hata değeri ve seçilme sıklığı için kullanılan e_2 hata değerinin, standart GA'nın bulduğu değerlere göre daha iyi olduğu görülmektedir.

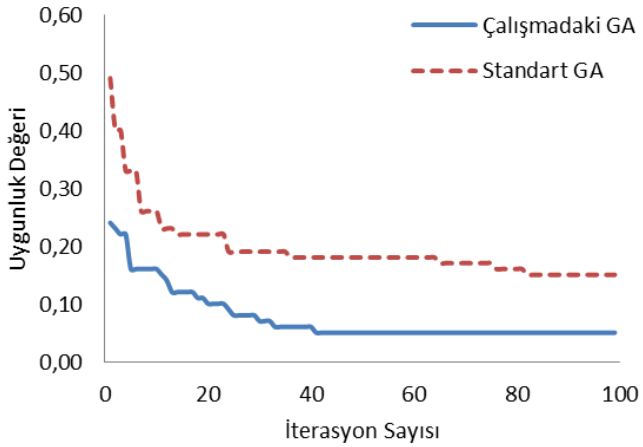
```
--- RESULTS For Improved GA --------------
Difficulty:0.0 | avgKnwPoint:2.041 | avgSelection:0.84 | totalAnswerTime:100.0 | e1:0.0 | e2:0.041 | e3:0.084 | e4:0.0 | FITNESS:0.031
Question IDs            : 20|96|127|212|226|286|357|370|416|494|594|597|635|652|664|778|831|903|923|925|1080|1151|1432|1442|1449|1460|
1466|1475|1525|1610|1634|1662|1706|1729|1773|1863|1870|1928|2013|2044|2102|2107|2115|2119|2124|2138|2148|2176|2182|2191|
Question Difficulties   : 1 3 4 3 2 2 2 3 3 2 4 4 4 4 4 3 3 4 3 3 3 2 4 3 2 3 4 2 2 2 3 4 4 4 2 2 3 3 3 4 3 4 4 4 4 3 1 3 3 3 3 3
Question Know. Points   : 1 2 1 2 2 2 2 2 3 1 2 3 3 3 3 3 3 1 1 1 1 2 1 3 1 2 1 1 2 2 1 3 3 3 3 3 1 2 1 3 3 1 3 3 3 3 2 2 1 2 2 2 2 2
Question Prev. Selected : 0 1 3 0 0 1 1 0 1 0 0 1 1 1 0 0 0 1 2 1 0 3 0 0 0 3 1 1 0 3 3 0 1 0 1 3 0 1 1 2 1 0 4 0 0 0 0 0 0 0
Question Answer Times   : 1 2 1 2 2 2 2 2 3 1 2 3 3 3 3 3 3 1 1 1 1 2 1 3 1 2 1 1 2 2 1 3 3 3 3 3 1 2 1 3 3 1 3 3 3 3 2 2 1 2 2 2 2 2
Question Chapters       : 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5
Question Scores         : 1 2 1 2 2 2 2 3 1 2 3 3 3 3 3 3 1 1 1 1 2 1 3 1 2 1 1 2 2 1 3 3 3 3 3 1 2 1 3 3 1 3 3 3 3 2 2 1 2 2 2 2 2
Total Prev. Selected Q. : 42.0


--- RESULTS For Traditional GA --------------
Difficulty:0.3 | avgKnwPoint:2.041 | avgSelection:1.56 | totalAnswerTime:100.0 | e1:0.3 | e2:0.041 | e3:0.156 | e4:0.0 | FITNESS:0.124
Question IDs            : 11|90|144|153|172|311|321|371|395|424|540|542|593|596|602|612|632|725|748|894|1143|1154|1195|1251|1275|1360|
1366|1371|1572|1622|1645|1648|1656|1660|1719|1783|1874|1878|1955|2052|2089|2100|2108|2177|2183|2216|2299|2302|2324|2448|
Question Difficulties   : 2 3 2 2 2 3 1 3 4 3 1 1 4 4 4 4 4 3 2 1 1 4 2 3 3 1 3 4 2 2 1 4 2 4 4 1 3 3 4 3 4 4 4 4 4 3 3 3 1 1 4 1
Question Know. Points   : 1 3 2 2 1 1 1 1 3 1 1 1 3 3 3 3 3 3 2 1 1 3 2 2 3 1 3 1 2 2 1 3 2 3 3 1 2 2 3 3 2 3 3 3 3 2 2 2 1 1 1 1
Question Prev. Selected : 3 2 1 1 4 0 0 1 1 0 3 0 2 0 1 3 2 0 2 1 4 2 3 3 0 2 4 1 2 2 4 1 1 4 1 0 0 0 0 1 3 2 0 1 3 4 2 0 0 1
Question Answer Times   : 1 3 2 2 1 1 1 1 3 1 1 1 3 3 3 3 3 3 2 1 1 3 2 2 3 1 3 1 2 2 1 3 2 3 3 1 2 2 3 3 2 3 3 3 3 2 2 2 1 1 1 1
Question Chapters       : 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5
Question Scores         : 1 3 2 2 1 1 1 1 3 1 1 1 3 3 3 3 3 3 2 1 1 3 2 2 3 1 3 1 2 2 1 3 2 3 3 1 2 2 3 3 2 3 3 3 3 2 2 2 1 1 1 1
Total Prev. Selected Q. : 78.0
```

Şekil 5.11 : Her kriterin ağırlık çarpanı %25 olan sonuç örneği.

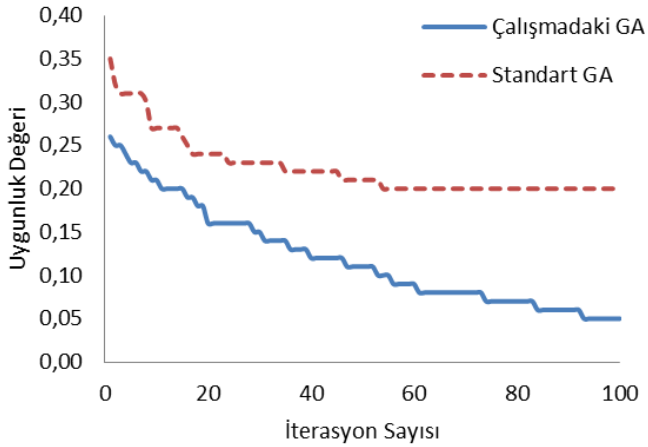
Şekil 5.12, Şekil 5.13, Şekil 5.14 ve Şekil 5.15'de, zorluk seviyesi 3, iterasyon sayısı 100 ve nüfus sayısı 80 olarak belirlenmiş ve soru sayısı farklı olan test sayfaları üretiminde standart GA ile çalışmadaki GA uygunluk değerlerinin iterasyona göre yakınsama grafikleri gösterilmektedir.



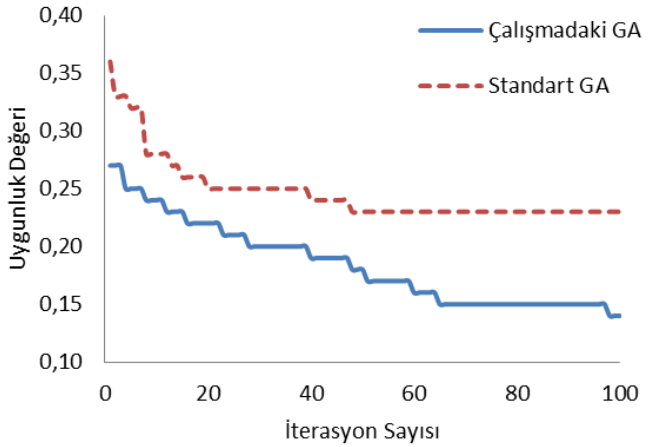
Şekil 5.12 : 20 soruluk test sayfası için karşılaştırmalı yakınsama grafiği.



Şekil 5.13 : 50 soruluk test sayfası için karşılaştırmalı yakınsama grafiği.

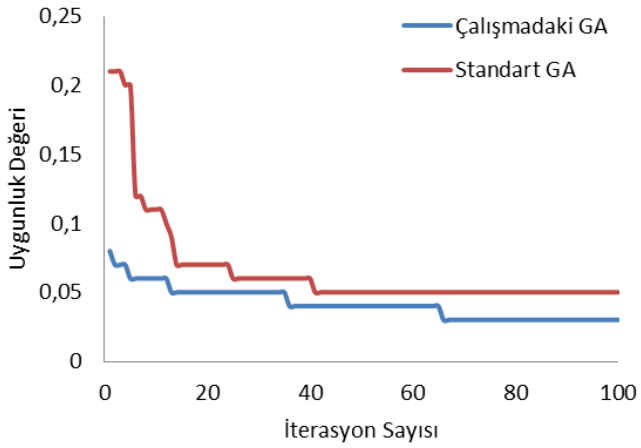


Şekil 5.14 : 100 soruluk test sayfası için karşılaştırmalı yakınsama grafiği.

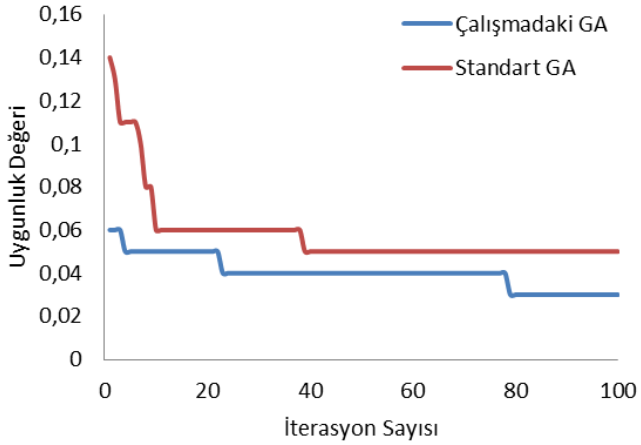


Şekil 5.15 : 200 soruluk test sayfası için karşılaştırmalı yakınsama grafiği.

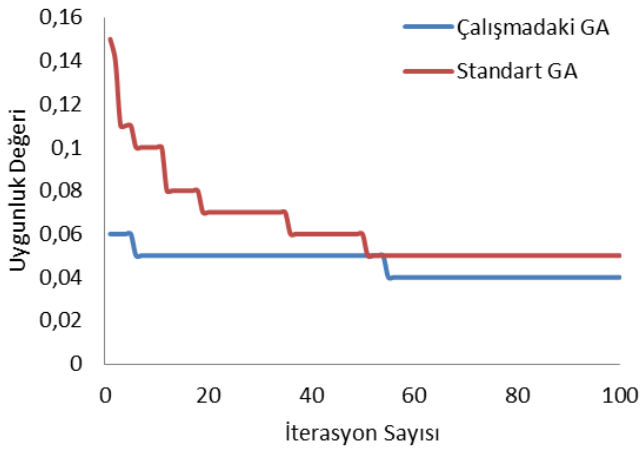
Şekil 5.16, Şekil 5.17, Şekil 5.18 ve Şekil 5.19'da, zorluk seviyesi 2, iterasyon sayısı 100 ve nüfus sayısı 80 olarak belirlenmiş ve soru sayısı farklı olan test sayfaları üretiminde standart GA ile çalışmadaki GA uygunluk değerlerinin iterasyona göre yakınsama grafikleri gösterilmektedir.



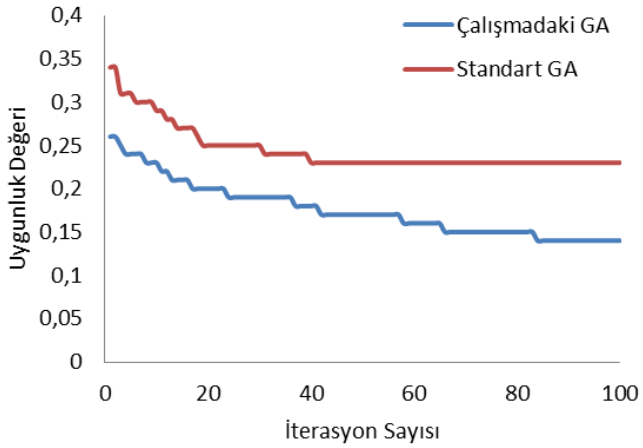
Şekil 5.16 : 20 soruluk test sayfası için karşılaştırmalı yakınsama grafiği.



Şekil 5.17 : 50 soruluk test sayfası için karşılaştırmalı yakınsama grafiği.

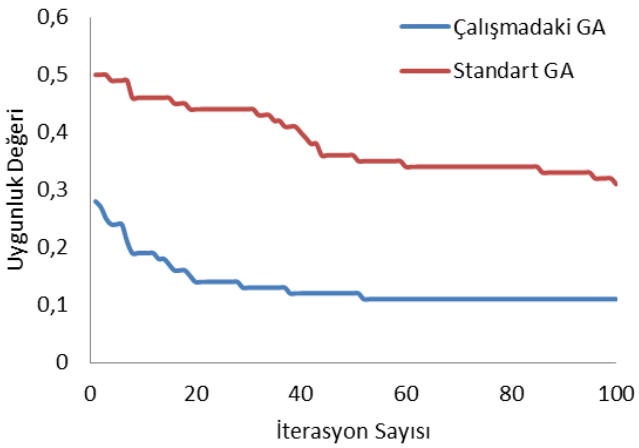


Şekil 5.18 : 100 soruluk test sayfası için karşılaştırmalı yakınsama grafiği.

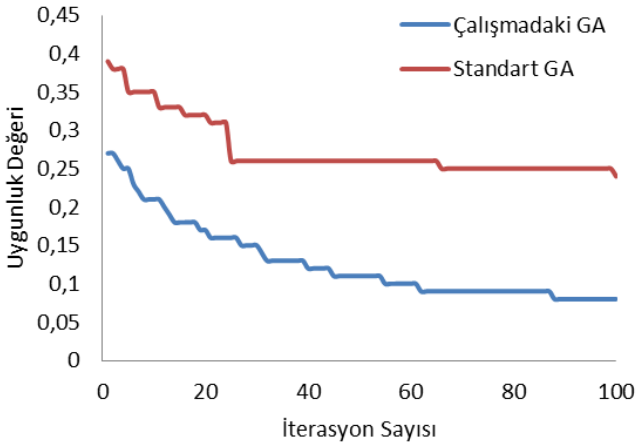


Şekil 5.19 : 200 soruluk test sayfası için karşılaştırmalı yakınsama grafiği.

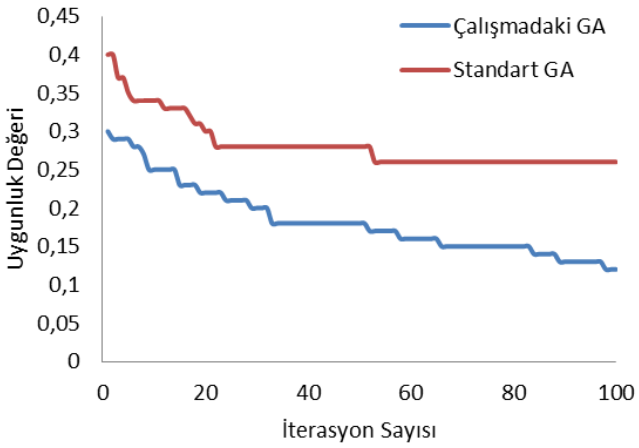
Şekil 5.20, Şekil 5.21, Şekil 5.22 ve Şekil 5.23'de, zorluk seviyesi 1, iterasyon sayısı 100 ve nüfus sayısı 80 olarak belirlenmiş ve soru sayısı farklı olan test sayfaları üretiminde standart GA ile çalışmadaki GA uygunluk değerlerinin iterasyona göre değişim grafikleri gösterilmektedir.



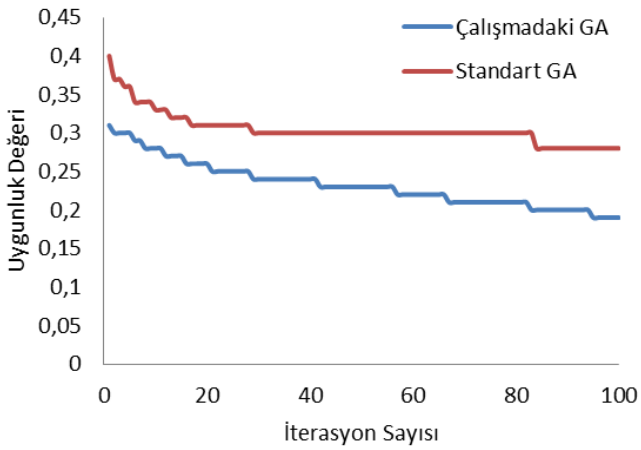
Şekil 5.20 : 20 soruluk test sayfası için karşılaştırmalı yakınsama grafiği.



Şekil 5.21 : 50 soruluk test sayfası için karşılaştırmalı yakınsama grafiği.



Şekil 5.22 : 100 soruluk test sayfası için karşılaştırmalı yakınsama grafiği.



Şekil 5.23 : 200 soruluk test sayfası için karşılaştırmalı yakınsama grafiği.

6. SONUÇ VE ÖNERİLER

Bu tez çalışmasında, test sayfası oluşturma konusu üzerinde durulmuş ve GA tabanlı bir çözüm yöntemi sunulmuştur. İnsan faktörüne bağlı olabilecek hataları azaltarak zamandan ve iş gücünden kazanç sağlayan bu yöntemle etkin bir şekilde test sayfaları oluşturulabilmektedir.

Literatür çalışmaları değerlendirilerek, çaprazlama ve mutasyon işlemleri standart GA'dan farklı olarak bölüm bazlı seçilen sorular üzerinden uygulanmıştır. Test soruları oluşturma işleminde sorular için zorluk seviyesi, bilgi puanı, cevaplama süresi ve seçilme sıklığı gibi dört farklı kriter kullanılarak, hazırlanan test sorularının daha etkin ve verimli olması sağlanmıştır. Yapılan deneysel çalışmalarda, önerilen GA ile standart GA sonuçları karşılaştırılmış ve önerilen algoritmanın daha başarılı sonuçlar verdiği görülmüştür.

İstenen kriterde test sorularının üretimi için kullanıcıların da kolay bir şekilde kullanabilecekleri web tabanlı bir kullanıcı arayüzü tasarımı gerçekleştirilmiştir. Web tabanlı kullanıcı arayüzü ile GA'nın parametreleri, test sayfası için istenilen özellikler ve özelliklerin ağırlıkları kullanıcı tarafından belirlenebilmekte ve buna göre istenilen özelliklere sahip veya en yakın özellikte test sayfaları oluşturulabilmektedir. Uygulama için Java, SpringBoot teknolojileri ve Thymeleaf gibi template yapılar kullanılmıştır. Veri tabanı olarak MongoDB (noSql) veri tabanı tercih edilirken arayüz için MongoBooster aracı kullanılmıştır.

Çalışmadaki bu yöntem, kullanımı gittikçe artan ve gelecekte klasik sınavların yerini alacağı düşünülen e-sınav sisteminde kullanılabileceği gibi soru bankası elektronik ortamda olan klasik sınav ve testler için de kullanılabilir.

Bu tez çalışmasının sonraki aşamalarında sorulara ait farklı özelliklerin belirlenip amaç fonksiyonunda kullanılmasıyla ya da nüfusun o özelliklere bağlı kalarak oluşturulup değiştirilmesiyle daha detaylı test sayfalarının hazırlanması mümkün olabilecektir. Çaprazlama ve mutasyon işlemleri, nüfusun anlık durumu dikkate alınarak belirlenebilecek olasılık formülleri ile gerçekleştirilerek algoritmanın daha da iyileştirilmesi sağlanabilir.

KAYNAKLAR

Ackley, D. H., 1987: A Connectionist Machine for Genetic Hillclimbing. Kluver Academic Publishers, Boston.

Akay, B., 2009: Nümerik Optimizasyon Problemlerinde Yapay Arı Kolonisi (Artificial Bee Colony) Algoritmasının Performans Analizi, Doktora Tezi, Erciyes Üniversitesi, Fen Bilimleri Enstitüsü, Kayseri.

Akşam, M. İ., 2014: Parçacık Sürü Optimizasyonu ile E-Sınav Uygulaması, Yüksek Lisans Tezi, Gazi Üniversitesi, Bilişim Enstitüsü, Ankara.

Anand, V., Spears, W. M., 1991: A Study Of Crossover Operators in Genetic Programming. The 6th International Symposium on Methodologies for Intelligent Systems. Charlotte, N:C, USA.

Beyazşekeroğlu, Ü., 2015: Moodle Öğrenme Yönetim Sistemi Üzerinde Matlab Yazılımı Kullanılarak Akıllı Soru Bankası Gerçekleştirilmesi, Yüksek Lisans Tezi, Kocaeli Üniversitesi, Fen Bilimleri Enstitüsü, Kocaeli.

Bhirangi, R., Bhoir, S., 2016: Automated Question Paper Generation System, International Journal of Emerging Research in Management & Technology ISSN: 2278-9359 (Volume-5, Issue-4), IEEE.

Cheng, S. C., Lin, Y. T., & Huang, Y. M., 2009: Dynamic question generation system for web-based testing using particle swarm optimization. Expert systems with applications, 36(1), 616-624.

Deb, K., Pratap, A., Agarwal, S. & Meyarivan, T., 2002: A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197.

Dilaver, D., 2015: Genetik algoritmalar yardımıyla iş atölye çizelgelemesi üzerine bir çalışma, Yüksek Lisans Tezi, Dokuz Eylül Üniversitesi, Sosyal Bilimler Enstitüsü, İzmir.

Dorigo, M., Gambardella, L. M., 1997: Ant Colonies for the Travelling Salesman Problem, Biosystems, 43 (2), 73–81.

Emir, Ş., 2006: E-öğrenmede sınav modelleri ve uygulaması, Yüksek Lisans Tezi, İstanbul Üniversitesi, Fen Bilimleri Enstitüsü, İstanbul.

Er, H. Ş., 2013: Gezgin satıcı probleminin hadoop üzerinde çalışan paralel genetik algoritma ile çözümü, Yüksek Lisans Tezi, İstanbul Teknik Üniversitesi, Fen Bilimleri Enstitüsü, İstanbul.

Erdal, M., 2007: Kısıtlı Kaynak Koşullarında Yapı Projelerinin Genetik Algoritma ile Programlanması, Doktora Tezi, Gazi Üniversitesi, Fen Bilimleri Enstitüsü, Ankara.

Gen, M., Cheng, R., 1997: Genetic Algorithms and Engineering Desing. John Wiley & Sons, Inc. ISBN 0-471-12741-8.

Genetic Algorithm, (t.y.). In Wikipedia. Alındığı tarih: 06.09.2016 http://en.wikipedia.org/wiki/Genetic_algorithm.

Glover, F., 1989: Tabu Search – Part I Part . ORSA Journal on Computing, Vol. 1, No. 3, pp 190-206.

Glover, F., Laguna, M., 1997: Tabu Search, Kluwer Academic Publishers, 61.

Goldberg, D. E., 1983: Computer-aided gas pipeline operation using genetic algorithms and rule learning, PhD thesis. University of Michigan. Ann Arbor, MI.

Goldberg, D. E., 1989: Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Publishing Company.

Gülcü, A., 2006: Yapay zeka tekniklerinden genetik algoritma ve tabu arama yöntemlerinin eğitim kurumlarının haftalık ders programlarının hazırlanmasında kullanımı, Yüksek Lisans Tezi, Marmara Üniversitesi, Fen Bilimleri Enstitüsü, İstanbul.

Hairui, W., & Hua, W., 2008: Research and implementation of multi-agent based test paper generation algorithm. In Computer Science and Software Engineering, 2008 International Conference on (Vol. 1, pp. 493-496), IEEE.

Holland, J. H., 1992: Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, Michigan; re-issued by MIT Press.

Jia, Z. H., Zhang, C. E., & Fang, H. S., 2011: The research and application of general item bank automatic test paper generation based on improved genetic algorithms. In Computing, Control and Industrial Engineering (CCIE), 2011 IEEE 2nd International Conference on (Vol. 1, pp. 14-18), IEEE.

Jun, N., 2014: An improved genetic algorithm for Intelligent test paper generation, Intelligent Computation Technology and Automation (ICICTA), 7th International Conference on IEEE, 72-75, IEEE.

Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., 1983: Optimization by Simulated Annealing, Science, 220, ss. 671–680.

Karataş, Ş., 2009: Akıllı e-Soru Sınav Sistemi Tasarımı ve Uygulaması, Yüksek Lisans Tezi, Beykent Üniversitesi, Fen Bilimleri Enstitüsü, İstanbul.

Li, G., Yu, L., & Sun, H., 2016: A framework for test data generation of object-oriented programs based on complete testing chain. In Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2016 17th IEEE/ACIS International Conference on (pp. 391-397), IEEE.

Melanie, M., 1999: An Introduction to Genetic Algorithms (5. baskı), Bradford Book The MIT Press, Cambridge, Massachusetts, London, England, 978-0-262-13316-6.

Maniezzo, V., Gambardella L. M., De Luigi, F., 2004: Ant Colony Optimization, New optimization techniques in engineering, Springer, Heidelberg, Germany, 101–121.

Ming-Zhu, S., Wei-Feng, L., & Jing-Yi, D., 2013: The Research and Implementation of Technology of Generating Test Paper Based on Genetic Algorithm. In Intelligence Computation and Evolutionary Computation (pp. 657-663). Springer, Berlin, Heidelberg, IEEE.

Nabiyev, V. V., 2005: Yapay Zeka Problemler, Yöntemler Ve Algoritmalar. Seçkin Yayıncılık.

Nguyen M. L., Hui S. C., Fong A. C., 2011: An efficient multi-objective optimization approach for online test paper generation, In Computational Intelligence in Multicriteria Decision-Making (MDCM), 2011 IEEE Symposium on (pp. 182-189), IEEE.

Özkan, R., 2003: Tek Modelli Deterministik Montaj Hattı Dengelem Problemlerine Genetik Algoritma ile Çözüm Yaklaşımı. Yüksek Lisans Tezi, İstanbul Teknik Üniversitesi, Fen Bilimleri Enstitüsü, İstanbul.

Razali, N. M., & Geraghty, J., 2011: Genetic algorithm performance with different selection strategies in solving TSP. In Proceedings of the world congress on engineering (Vol. 2, pp. 1134-1139). Hong Kong: International Association of Engineers.

Reeves, C. R., 1993, Modern Heuristic Techniques for Combinatorial Problems, Halsted Press, John Wiley & Sons. Inc., New York, 303s.

Sel, Ç., 2013: Genetik Atama Problemlerinin Çözümünde Deterministik, Olasılık Temelli ve Sezgisel Yöntemlerin Uygulanması, Yüksek Lisans Tezi, Ankara Üniversitesi, Fen Bilimleri Enstitüsü, Ankara.

Shan, Y., 2010 : The Research and Realization of Multi-threaded Intelligent Test Paper Generation Based on Genetic Algorithm, International Conference on Computer and Information Application (ICCIA), 461-464, IEEE.

Sun, X., 2009: Study on Test Databank Construction And Algorithm of Test Paper Generation System, Second International Symposium on Electronic Commerce and Security (ISECS), 297-302, IEEE.

Tang, K. S., Man, K. F., Kwong, S. & He, Q., 1996: Genetic Algorithms and Their Applications, IEEE Signal Processing Magazine, vol. 13, no. 6, pp. 22-37.

Torkul, O., Kibar, A., Taşcı, T., 2004: Web tabanlı sınav sistemleri, Sakarya Üniversitesi, Enformatik Bölüm Başkanlığı, 54187, Sakarya.

Tuncer, A., Yıldırım, M., 2012: Dynamic path planning of mobile robots with improved genetic algorithm, Computers & Electrical Engineering, Cilt 38, No 6, 1564-1572.

Tül, U., Tuncer, A., 2017: Genetik Algoritma ile Akıllı Test Sayfası Oluşturma. Gazi Üniversitesi Fen Bilimleri Dergisi Part C: Tasarım ve Teknoloji, 5 (4), 27-34. DOI: 10.29109, http-gujsc-gazi-edu-tr.341977.

Url-1, <http://www.cs.cmu.edu/~ark/QA-data/>, Carnegie Mellon University, Question-Answer Dataset, alındığı tarih: 01.08.2017.

Url-2, <http://web.firat.edu.tr/iaydin/bmu579/bmu_579_bolum6.pdf>, alındığı tarih: 04.12.2017.

Url-3, <http://w3.gazi.edu.tr/~akcayol/files/ZOL5Genetik.pdf> alındığı tarih: 17.12.2017.

Wu, X., Song, Y., 2009: Research on Intelligent Auto-generating Test Paper Based on Improved Genetic Algorithms, International Conference on Computational Intelligence and Software Engineering, International Conference on IEEE.

Xiong, L., Shi, J., 2010: Automatic Generating Test Paper System Based On Genetic Algorithm, Second International Workshop on Education Technology and Computer Science, IEEE.

Xiumin, C., Dengcai, W., Meining, Z., Yanping, Y., 2011: Research on Intelligent Test Paper Generation Base on Improved Genetic Algorithm, The 6th International Conference on Computer Science & Education (ICCSE), 269-272, IEEE.

Yang, R., 1997: Solving Large Travelling Salesman Problems with Small Populations, Department of Computer Science, University of Bristol, U.K.

Yıldırım, M., 2008: A genetic algorithm for generating test from a question bank. Computer Applications in Engineering Education, 18(2), 298-305.

Yong-kang, P., Wang-ren, Q., 2011: Intelligent test paper generation research based on the interval-valued fuzzy theory. In System Science, Engineering Design and Manufacturing Informatization (ICSEM), 2011 International Conference on (Vol. 1, pp. 271-274), IEEE.

Zhang, K., Zhu, L., 2015: Application of Improved Genetic Algorithm in Automatic Test Paper Generation, Chinese Automation Congress (CAC), 495-499, IEEE.

Zhong, R. W., Wang, H. P., 2010: Genetic Algorithm in Test Paper Generation System, In E-Product E-Service and E-Entertainment (ICEEE), 2010 International Conference on (pp. 1-4), IEEE.

EKLER

EK A.1 : Uygulama Kaynak Kodları (Sınıf ve metotların bazılarını içerir)

EK A.1

```
package testpaper.generation.model;

import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "questions")
public class Question {

	private String id;

	private Integer questionId;

	private String articleTitle;

	private String text;

	private String answer;

	private String articleFile;

	private Integer chapter;

	private Integer knowledgePoint;

	private Integer difficulty;

	private String difficultyFromQuestioner;

	private String difficultyFromAnswerer;

	private Integer estimatedAnswerTime;

	private Integer score;

	private Integer selectionRatio;

	public String getId() {
		return id;
	}

	public void setId(String id) {
		this.id = id;
	}

	public String getArticleTitle() {
		return articleTitle;
	}

	public void setArticleTitle(String articleTitle) {
		this.articleTitle = articleTitle;
	}

	public String getText() {
		return text;
	}

	public void setText(String text) {
		this.text = text;
	}
```

```
public Integer getScore() {
        return score / 10;
}

public void setScore(Integer score) {
        this.score = score;
}

public String getAnswer() {
        return answer;
}

public void setAnswer(String answer) {
        this.answer = answer;
}

public String getArticleFile() {
        return articleFile;
}

public void setArticleFile(String articleFile) {
        this.articleFile = articleFile;
}

public Integer getChapter() {
        return chapter;
}

public void setChapter(Integer chapter) {
        this.chapter = chapter;
}

public Integer getKnowledgePoint() {
        return knowledgePoint;
}

public void setKnowledgePoint(Integer knowledgePoint) {
        this.knowledgePoint = knowledgePoint;
}

public Integer getDifficulty() {
        return difficulty;
}

public void setDifficulty(Integer difficulty) {
        this.difficulty = difficulty;
}

public Integer getEstimatedAnswerTime() {
        return estimatedAnswerTime;
}

public void setEstimatedAnswerTime(Integer estimatedAnswerTime) {
        this.estimatedAnswerTime = estimatedAnswerTime;
}

public Integer getQuestionId() {
        return questionId;
}
```

```
	public void setQuestionId(Integer questionId) {
		this.questionId = questionId;
	}

	public String getDifficultyFromQuestioner() {
		return difficultyFromQuestioner;
	}

	public void setDifficultyFromQuestioner(String difficultyFromQuestioner) {
		this.difficultyFromQuestioner = difficultyFromQuestioner;
	}

	public String getDifficultyFromAnswerer() {
		return difficultyFromAnswerer;
	}

	public void setDifficultyFromAnswerer(String difficultyFromAnswerer) {
		this.difficultyFromAnswerer = difficultyFromAnswerer;
	}

	public Integer getSelectionRatio() {
		return selectionRatio;
	}

	public void setSelectionRatio(Integer selectionRatio) {
		this.selectionRatio = selectionRatio;
	}

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((questionId == null) ? 0 : questionId.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Question other = (Question) obj;
		if (questionId == null) {
			if (other.questionId != null)
				return false;
		} else if (!questionId.equals(other.questionId))
			return false;
		return true;
	}

	@Override
	public String toString(){
		return "" + this.questionId;
	}
}
```

```java
package testpaper.GA;

import testpaper.generation.model.Properties;

public class Population implements Cloneable {

    public TestPaper[] testPapers;

    private Properties properties;

    private TestPaper  fittest;

    /***
     * Constructor, verilen özellikler bilgisine göre populasyon oluşturur.
     *
     * @param populationSize
     * @param initialise
     */
    public Population(final Properties properties) {
        this.properties = properties;
        this.testPapers = new TestPaper[properties.getPOPULATION_SIZE()];
    }

    /***
     * Verilen indexdeki test sayfasını doner
     *
     * @param index
     * @return
     */
    public TestPaper getTestPaper(final int index) {
        return this.testPapers[index];
    }

    /***
     * Populasyondaki en iyi test sayfasını verir
     *
     * @return
     */
    public TestPaper getFittest() {
        this.fittest = this.testPapers[0];
        for (int i = 0; i < this.size(); i++) {
            if (this.getTestPaper(i).getFitnessValue() < this.fittest.getFitnessValue()) {
                this.fittest = this.getTestPaper(i);
            }
        }
        return this.fittest;
    }

    /***
     * Populasyon size bilgisini verir
     *
     * @return
     */
    public int size() {
        return this.testPapers.length;
    }

    /***
     * Verilen indexe göre verilen test sayfasını atar
```

```
     *
     * @param index
     * @param testPaper
     */
    public void saveTestPaper(final int index, final TestPaper testPaper) {
        this.testPapers[index] = testPaper;
    }

    public Properties getProperties() {
        return this.properties;
    }

    public void setProperties(final Properties properties) {
        this.properties = properties;
    }

    @Override
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}
```

```
package testpaper.GA;

import java.util.ArrayList;
import java.util.List;

import testpaper.generation.controllers.Algorithm;
import testpaper.generation.model.Properties;
import testpaper.generation.model.Question;

/**
 * The Class TestPaper.
 */
public class TestPaper {

    public List<Question> questionList;

    private int          punishment;

    private double       difficulty            = 0.0;

    private double       knowledgePoint        = 0.0;

    private double       score                 = 0.0;

    private double       estimatedAnswerTime   = 0.0;

    private double       chapterCompliance     = 0.0;

    private double       ui                    = 0.0;

    private double       fitnessValue          = 0.0;

    private double       difficultyCoefficient_W = 0.0;

    private double       totalScore_S          = 0.0;

    private double       knowledgePoint_Q      = 0.0;
```

```
    private double      totalTime_T         = 0.0;

    private double      selectionRatio      = 0.0;

    private Properties   properties;

    private double      weight1;

    private double      weight2;

    private double      weight3;

    private double      weight4;

    /**
     * Instantiates a new testPaper.
     */
    public TestPaper(final Properties properties) {
        this.questionList = new ArrayList<Question>();
        this.properties = properties;
        this.calculateWeights();
    }

    public Question getQuestion(final int index) {
        return this.questionList.get(index);
    }

    public void setQuestion(final int index, final Question question) {
        this.questionList.set(index, question);
    }

    public int getPunishment() {
        this.punishment = FitnessCalc.getPunishment(this);
        return this.punishment;
    }

    public void calculateWeights() {
        final int totalW = this.getProperties().getW1() + this.getProperties().getW2() +
this.getProperties().getW3() + this.getProperties().getW4();
        this.weight1 = (double) this.getProperties().getW1() / totalW;
        this.weight2 = (double) this.getProperties().getW2() / totalW;
        this.weight3 = (double) this.getProperties().getW3() / totalW;
        this.weight4 = (double) this.getProperties().getW4() / totalW;
    }

    public void sortByQuestionID() {
        this.questionList.sort((p1, p2) -> p1.getQuestionId().compareTo(p2.getQuestionId()));
    }

    public String getQuestionsAsString() {
        String value = "";
        for (final Question q : this.questionList) {
            value += q + " | ";
        }
        return value;
    }

    public String getDifficultiesAsString() {
        String value = "";
```

```
    for (final Question q : this.questionList) {
        value += " " + q.getDifficulty() + " - ";
    }
    return value;
}

public String getScoresAsString() {
    String value = "";
    for (final Question q : this.questionList) {
        value += q.getScore() + " - ";
    }
    return value;
}

public String getSelectionRatioAsString() {
    String value = "";
    for (final Question q : this.questionList) {
        value += q.getSelectionRatio() + " - ";
    }
    return value;
}

public String getKnowledgePointsAsString() {
    String value = "";
    for (final Question q : this.questionList) {
        value += q.getKnowledgePoint() + " - ";
    }
    return value;
}

public String getGeneScoreAsString() {
    String value = "";
    for (final Question q : this.questionList) {
        value += q.getScore() + " - ";
    }
    return value;
}

public String getQuestionListAsString() {
    String value = "";
    for (final Question q : this.questionList) {
        value += q.getQuestionId() + " - ";
    }
    return value;
}

public String getQuestionChaptersAsString() {
    String value = "";
    for (final Question q : this.questionList) {
        value += q.getChapter() + " - ";
    }
    return value;
}

public String getQuestionAnswerTimesAsString() {
    String value = "";
    for (final Question q : this.questionList) {
        value += q.getEstimatedAnswerTime() + " - ";
    }
    return value;
```

```
}

/***
 * TestPaper zorluk derecesi farkını (genleri değiştiği için her çağrımında) hesaplayarak tutar.
 *
 * Zorluk derecesi farkını #.## şeklinde round ederek mutlak değerini alır.
 *
 * @return testin zorluk derecesi farkı
 */
public double getDifficulty() {
   this.difficulty = FitnessCalc.getDifficulty(this);
   this.difficulty = Math.abs(this.round(this.difficulty));
   return this.difficulty;
}

public void setDifficulty(final double difficulty) {
   this.difficulty = difficulty;
}

public double getKnowledgePoint() {
   this.knowledgePoint = 0.0;
   for (final Question q : this.questionList) {
      this.knowledgePoint += q.getKnowledgePoint();
   }
   return this.knowledgePoint;
}

public double getSelectionRatio() {
   this.selectionRatio = 0.0;
   for (final Question q : this.questionList) {
      this.selectionRatio += q.getSelectionRatio();
   }
   return this.selectionRatio;
}

public double getScore() {
   this.score = 0.0;
   for (final Question q : this.questionList) {
      this.score += q.getScore();
   }
   return this.score;
}

public double getEstimatedAnswerTime() {
   this.estimatedAnswerTime = 0.0;
   for (final Question q : this.questionList) {
      this.estimatedAnswerTime += q.getEstimatedAnswerTime();
   }
   return this.estimatedAnswerTime;
}

public double getE1() {
   final double e = this.getDifficulty();
   return this.round(e);
}

public double getE2() {
   final double e = Math.abs(FitnessCalc.desiredKnowledgePoint - this.getKnowledgePointAvg());
   return this.round(e);
}
```

```
    public double getE3() {
        final double e = this.getSRatio();
        return this.round(e);
    }

    public double getE4() {
        final double e = Math.abs(FitnessCalc.desiredAnswerTime - this.getEstimatedAnswerTime()) /
this.getProperties().getDEFAULT_GENE_LENGTH();
        return this.round(e);
    }

    public double getSRatio() {
        final double sr = this.getSelectionRatio() /
(this.getProperties().getDEFAULT_GENE_LENGTH());
        return this.round(sr);
    }

    public double round(double value) {
        value = Math.round(value * 100);
        value = value / 100;
        return value;
    }

    public double getFitnessValue() {
        this.fitnessValue = (this.getE1() * this.weight1) // Difficulty
                + (this.getE2() * this.weight2) // Knowledge Point
                + (this.getE3() * this.weight3) // Selection Ratio
                + (this.getE4() * this.weight4) // Answer Time
                + Algorithm.getPunismentValue(this);

        return this.round(this.fitnessValue);
    }

    public double getUi() {
        return this.ui;
    }

    public void setUi(final double ui) {
        this.ui = ui;
    }

    public String toDetailString() {
        return super.toString()
                + " diff:"
                + this.getDifficulty()
                + ", AvgScore:"
                + this.getScoreAvg()
                + ", AvgKnwPoint:"
                + this.getKnowledgePointAvg()
                + ", E1:"
                + this.getE1()
                + ", E2:"
                + this.getE2()
                + ", E3:"
                + this.getE3()
                + ", E4:"
                + this.getE4()
                + ", FTN:"
                + this.getFitnessValue();
```

```
}

public String toResultString() {
   return "tekrarli soru:" + this.getPunishment() + " | zorluk farki:" + this.getDifficulty();
}

public double getDifficultyCoefficient_W() {
   double temp = 0.0;
   for (final Question q : this.questionList) {
      temp += q.getScore() * q.getDifficulty();
   }
   this.difficultyCoefficient_W = temp / this.getTotalScore_S();
   return this.round(this.difficultyCoefficient_W);
}

public void setDifficultyCoefficient_W(final double difficultyCoefficient_W) {
   this.difficultyCoefficient_W = difficultyCoefficient_W;
}

public double getTotalScore_S() {
   this.totalScore_S = this.getScore();
   return this.totalScore_S;
}

public double getScoreAvg() {
   return this.round(this.getTotalScore_S() / this.questionList.size());
}

public double getDesiredScoreAvg() {
   return this.round(FitnessCalc.desiredScore / this.questionList.size());
}

public double getKnowledgePointAvg() {
   for (final Question q : this.questionList) {
      this.knowledgePoint_Q += q.getKnowledgePoint();
   }
   this.knowledgePoint_Q = this.knowledgePoint_Q / this.questionList.size();
   return this.round(this.knowledgePoint_Q);
}

public void setTotalScore_S(final double totalScore_S) {
   this.totalScore_S = totalScore_S;
}

public void setKnowledgePoint_Q(final double knowledgePoint_Q) {
   this.knowledgePoint_Q = knowledgePoint_Q;
}

public double getTotalTime_T() {
   this.totalTime_T = this.getEstimatedAnswerTime();
   return this.totalTime_T;
}

public void setTotalTime_T(final double totalTime_T) {
   this.totalTime_T = totalTime_T;
}

public void setKnowledgePoint(final double knowledgePoint) {
   this.knowledgePoint = knowledgePoint;
}
```

```
    public void setScore(final double score) {
        this.score = score;
    }

    public void setEstimatedAnswerTime(final double estimatedAnswerTime) {
        this.estimatedAnswerTime = estimatedAnswerTime;
    }

    public double getChapterCompliance() {
        return this.chapterCompliance;
    }

    public void setChapterCompliance(final double chapterCompliance) {
        this.chapterCompliance = chapterCompliance;
    }

    public void setPunishment(final int punishment) {
        this.punishment = punishment;
    }

    public Properties getProperties() {
        return this.properties;
    }

    public void setProperties(final Properties properties) {
        this.properties = properties;
    }

    public void setSelectionRatio(final double selectionRatio) {
        this.selectionRatio = selectionRatio;
    }

    public List<Question> getQuestionList() {
        return this.questionList;
    }

    public void setQuestionList(final List<Question> questionList) {
        this.questionList = questionList;
    }
}

package testpaper.generation.model;

import java.util.Comparator;
import java.util.Random;

import testpaper.GA.TestPaper;

/**
 * The Class Constant.
 */
public class Properties {

    /** The Constant DEFAULT_GENE_LENGTH. */
    private int          DEFAULT_GENE_LENGTH   = 50;

    /** The Constant POPULATION SIZE */
    private int          POPULATION_SIZE       = 50;
```

```
/** The Constant POOL_SIZE. */
private int          STATE_SPACE          = 2455;

/** The Constant MAX_ITERATION. */
private int          MAX_ITERATION        = 100;

/** The Constant TEST DIFFICULTY. */
private int          DIFFICULTY           = 3;

/** The Constant TEST SCORE. */
private int          SCORE                = 100;

/** The Constant KNOWLEDGE_POINT. */
private int          KNOWLEDGE_POINT      = 2;

/** The Constant ESTIMATED_ANSWER_TIME. */
private int          ESTIMATED_ANSWER_TIME = this.DEFAULT_GENE_LENGTH * 2;

/** Uygunluk fonksiyonu MAX degeri */
public static final int U_MAX             = 1;

private int          chapter1             = 10;

private int          chapter2             = 10;

private int          chapter3             = 10;

private int          chapter4             = 10;

private int          chapter5             = 10;

private boolean      excludeLastQuestions;

private int          w1                   = 1, w2 = 1, w3 = 1, w4 = 1;

private boolean      improvedAlgorithm    = true;

/** The random. */
private static Random   random            = new Random();

/**
 * Instantiates a new statics.
 */
public Properties() {
}

public int getDEFAULT_GENE_LENGTH() {
   return this.DEFAULT_GENE_LENGTH;
}

public void setDEFAULT_GENE_LENGTH(final int dEFAULT_GENE_LENGTH) {
   this.DEFAULT_GENE_LENGTH = dEFAULT_GENE_LENGTH;
}

public int getPOPULATION_SIZE() {
   return this.POPULATION_SIZE;
}

public void setPOPULATION_SIZE(final int pOPULATION_SIZE) {
   this.POPULATION_SIZE = pOPULATION_SIZE;
```

```
}

public int getSTATE_SPACE() {
    return this.STATE_SPACE;
}

public void setSTATE_SPACE(final int sTATE_SPACE) {
    this.STATE_SPACE = sTATE_SPACE;
}

public int getMAX_ITERATION() {
    return this.MAX_ITERATION;
}

public void setMAX_ITERATION(final int mAX_ITERATION) {
    this.MAX_ITERATION = mAX_ITERATION;
}

public int getDIFFICULTY() {
    return this.DIFFICULTY;
}

public void setDIFFICULTY(final int dIFFICULTY) {
    this.DIFFICULTY = dIFFICULTY;
}

public Random getRandom() {
    return random;
}

public void setRandom(final Random random) {
    Properties.random = random;
}

public int getuMax() {
    return U_MAX;
}

public int getSCORE() {
    return this.SCORE;
}

public void setSCORE(final int SCORE) {
    this.SCORE = SCORE;
}

public int getKNOWLEDGE_POINT() {
    return this.KNOWLEDGE_POINT;
}

public void setKNOWLEDGE_POINT(final int kNOWLEDGE_POINT) {
    this.KNOWLEDGE_POINT = kNOWLEDGE_POINT;
}

public int getESTIMATED_ANSWER_TIME() {
    return this.ESTIMATED_ANSWER_TIME;
}

public void setESTIMATED_ANSWER_TIME(final int eSTIMATED_ANSWER_TIME) {
    this.ESTIMATED_ANSWER_TIME = eSTIMATED_ANSWER_TIME;
```

```
}

public int getU_MAX() {
    return U_MAX;
}

public int getChapter1() {
    return this.chapter1;
}

public void setChapter1(final int chapter1) {
    this.chapter1 = chapter1;
}

public int getChapter2() {
    return this.chapter2;
}

public void setChapter2(final int chapter2) {
    this.chapter2 = chapter2;
}

public int getChapter3() {
    return this.chapter3;
}

public void setChapter3(final int chapter3) {
    this.chapter3 = chapter3;
}

public int getChapter4() {
    return this.chapter4;
}

public void setChapter4(final int chapter4) {
    this.chapter4 = chapter4;
}

public int getChapter5() {
    return this.chapter5;
}

public void setChapter5(final int chapter5) {
    this.chapter5 = chapter5;
}

public int getW1() {
    return this.w1;
}

public void setW1(final int w1) {
    this.w1 = w1;
}

public int getW2() {
    return this.w2;
}

public void setW2(final int w2) {
    this.w2 = w2;
```

```
    }

    public int getW3() {
        return this.w3;
    }

    public void setW3(final int w3) {
        this.w3 = w3;
    }

    public int getW4() {
        return this.w4;
    }

    public void setW4(final int w4) {
        this.w4 = w4;
    }

    public boolean isExcludeLastQuestions() {
        return this.excludeLastQuestions;
    }

    public void setExcludeLastQuestions(final boolean excludeLastQuestions) {
        this.excludeLastQuestions = excludeLastQuestions;
    }

    public boolean isImprovedAlgorithm() {
        return this.improvedAlgorithm;
    }

    public void setImprovedAlgorithm(final boolean improvedAlgorithm) {
        this.improvedAlgorithm = improvedAlgorithm;
    }

    /***
     * Dizi ya da listeyi siralamak icin kullanilir. Test sayfası zorluk derecesini sıralamak icin.
     */
    public static final Comparator<TestPaper> testPaperComparator = new
Comparator<TestPaper>() {
        @Override
        public int compare(final TestPaper t1, final TestPaper t2) {
            return Double.compare(t1.getFitnessValue(), t2.getFitnessValue());
        }
    };

}


package testpaper.GA;

import testpaper.generation.model.Question;

public class FitnessCalc {

            static double desiredQuestionCount;

            static double desiredDifficulty;

            static double desiredScore;
```

```
	static double desiredKnowledgePoint;

	static double desiredAnswerTime;

	static double desiredChapterCompliance;

	/***
	 * Test sayfasinin kac tane tekrarli soruya sahip oldugunu doner
	 *
	 * @param testPaper
	 * @return
	 */
	static int getPunishment(TestPaper testPaper) {
		int punishment = 0;
		for (int i = 0; i < testPaper.questionList.size(); i++) {
			int count = 0;
			Question question = testPaper.getQuestion(i);
			for (Question q : testPaper.questionList) {
				if (q.getQuestionId() == question.getQuestionId()) {
					count++;
				}
			}
			punishment = punishment + count;
		}

		punishment = (punishment - testPaper.questionList.size()) / 2;
		return punishment;
	}


	/***
	 * Verilen Test sayfasının zorluk derecesini hesaplar, istenilen zorluk derecesiyle farkını
alarak döner.
	 *
	 * @param testPaper
	 * @return zorluk derecesi
	 */
	static double getDifficulty(TestPaper testPaper) {
		double difficulty = 0;
		for (Question q : testPaper.questionList) {
			difficulty = difficulty + q.getDifficulty();
		}
		double avgDifficulty = difficulty / testPaper.questionList.size();
		return avgDifficulty - getDesiredDifficulty();
	}


	/***
	 * Test sayfasinin hangi zorluk seviyesinde olacagini set eder
	 *
	 * @param solution
	 */
	public static void setDesiredDifficulty(double desiredDifficulty) {
		FitnessCalc.desiredDifficulty = desiredDifficulty;
	}

	/***
	 * Beklenen sonuç değeri
	 *
	 * @return
```

```
	 */
	static double getDesiredDifficulty() {
		return desiredDifficulty;
	}


	public static double getDesiredScore() {
		return desiredScore;
	}


	public static void setDesiredScore(double desiredScore) {
		FitnessCalc.desiredScore = desiredScore;
	}


	public static double getDesiredKnowledgePoint() {
		return desiredKnowledgePoint;
	}


	public static void setDesiredKnowledgePoint(double desiredKnowledgePoint) {
		FitnessCalc.desiredKnowledgePoint = desiredKnowledgePoint;
	}


	public static double getDesiredAnswerTime() {
		return desiredAnswerTime;
	}


	public static void setDesiredAnswerTime(double desiredAnswerTime) {
		FitnessCalc.desiredAnswerTime = desiredAnswerTime;
	}


	public static double getDesiredChapterCompliance() {
		return desiredChapterCompliance;
	}


	public static void setDesiredChapterCompliance(double desiredChapterCompliance) {
		FitnessCalc.desiredChapterCompliance = desiredChapterCompliance;
	}


	public static double getDesiredQuestionCount() {
		return desiredQuestionCount;
	}


	public static void setDesiredQuestionCount(double desiredQuestionCount) {
		FitnessCalc.desiredQuestionCount = desiredQuestionCount;
	}

}


package testpaper.generation.controllers;
```

```
import java.util.Arrays;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;

import testpaper.GA.Population;
import testpaper.GA.TestPaper;
import testpaper.generation.model.Properties;
import testpaper.generation.model.Question;
import testpaper.generation.repository.QuestionSearchRepository;

@Controller
public class Algorithm {

    @Autowired
    QuestionSearchRepository     questionSearchRepository;

    @Autowired
    QuestionGenerator           questionGenerator;

    @Autowired
    ConfigController            configController;

    // GA parameters
    private static final double  mutationRate = 0.015;
    private static final double  uniformRate  = 0.5;
    private static final boolean elitism      = true;

    /***
     * Populasyonu evolve eder, elit bireyi saklar, çaprazlama ve mutasyon methodlarını çağırır
     *
     * @param population
     * @return
     */
    public Population evolvePopulation(final Population population, final Properties properties) {
        final Population newPopulation = new Population(properties);

        if (elitism) {
            newPopulation.saveTestPaper(0, population.getFittest());
        }

        // Crossover
        int elitismOffset;
        if (elitism) {
            elitismOffset = 1;
        } else {
            elitismOffset = 0;
        }

        for (int i = elitismOffset; i < population.size(); i++) {
            final TestPaper t1 = rouletteWheelSelection(population);
            final TestPaper t2 = rouletteWheelSelection(population);

            TestPaper newTestPaper = null;
            if (properties.isChapterSelected() && properties.isImprovedAlgorithm()) {
                newTestPaper = this.crossoverByChapter(t1, t2, properties); // Chapter bazlı
            } else {
```

```
            newTestPaper = this.crossover(t1, t2, properties);
        }
        newPopulation.saveTestPaper(i, newTestPaper);
    }

    // Mutate population
    for (int i = elitismOffset; i < newPopulation.size(); i++) {
        this.mutate(newPopulation.getTestPaper(i), properties);
    }

    return newPopulation;
}

/***
 * Crossover ile testpaperlar içindeki questionlar çaprazlanır
 *
 * @param t1
 * @param t2
 * @return
 */
private TestPaper crossover(final TestPaper t1, final TestPaper t2, final Properties properties) {
    final TestPaper newTestPaper = new TestPaper(properties);
    final int crossPoint = randInt(0, properties.getDEFAULT_GENE_LENGTH());
    for (int i = 0; i < this.configController.properties.getDEFAULT_GENE_LENGTH(); i++) {
        if ((i < crossPoint) && !newTestPaper.questionList.contains(t1.getQuestion(i))) {
            newTestPaper.questionList.add(i, t1.getQuestion(i));
        } else if (!newTestPaper.questionList.contains(t2.getQuestion(i))) {
            newTestPaper.questionList.add(i, t2.getQuestion(i));
        } else {
            newTestPaper.questionList.add(i,
this.questionGenerator.getUniqueRandomQuestion(newTestPaper));
        }
    }
    return newTestPaper;
}

/***
 * Mutation ile test paperdaki question mutasyon oranına göre değiştirilebilir.
 *
 * @param testPaper
 */
private void mutate(final TestPaper testPaper, final Properties properties) {
    for (int i = 0; i < properties.getDEFAULT_GENE_LENGTH(); i++) {
        if (Math.random() <= mutationRate) {
            if (properties.isChapterSelected() && properties.isImprovedAlgorithm()) {
                final Question q =
this.questionGenerator.getRandomUniqueQuestionByChapter(testPaper.questionList.get(i).getChapte
r(), testPaper);
                testPaper.questionList.set(i, q);
            } else {
                testPaper.questionList.set(i, this.questionGenerator.getRandomQuestion());
            }
        }
    }
}

/***
 * Uygunluk fonksiyonu: bireyler arasinda iyi bireyden kotuye dogru siralama yapar ve uygunluk
degerlerini belirler
 *
```

```
     * @param population
     */
    public void sortAndCalculateFitness(final Population population) {
        System.setProperty("java.util.Arrays.useLegacyMergeSort", "true");
        Arrays.sort(population.testPapers, Properties.testPaperComparator); // f(x)'e göre değişir ***
        for (int i = 0; i < population.testPapers.length; i++) {
            final TestPaper testPaper = population.testPapers[i];
            final double ui = (Properties.U_MAX *
Double.valueOf((this.configController.properties.getPOPULATION_SIZE() - (i + 1))))
                    / (this.configController.properties.getPOPULATION_SIZE() - 1);
            testPaper.setUi(testPaper.round(ui));
        }

    }

    /***
     * Roullette Wheel yontemi ile secim
     *
     * @param population
     * @return
     */
    public static TestPaper rouletteWheelSelection(final Population population) {
        double range = 0;
        for (final TestPaper testPaper : population.testPapers) {
            range += testPaper.getUi();
        }

        final Random random = new Random();
        final double randomValue = random.nextDouble() * range;

        double sum = 0;
        for (int i = 0; i < population.testPapers.length; i++) {
            if (randomValue <= (population.testPapers[i].getUi() + sum)) {
                return population.testPapers[i];
            } else {
                sum += population.testPapers[i].getUi();
            }
        }
        return null;
    }

    /***
     * Ayni question varsa random question ile degistirir. Tekrarli sorulari onlemek amaclidir.
     *
     * @param testPaper
     */
    public void punishmentControl(final TestPaper testPaper, final Properties properties) {
        for (int i = 0; i < properties.getDEFAULT_GENE_LENGTH(); i++) {
            final Question currentQuestion = testPaper.getQuestion(i);
            int count = 0;
            for (final Question q : testPaper.questionList) {
                if (q.getId() == currentQuestion.getId()) {
                    count++;
                    if (count >= 2) {
                        // Create random question
                        if (properties.isChapterSelected() && properties.isImprovedAlgorithm()) {
                            testPaper.questionList.set(i,

this.questionGenerator.getRandomUniqueQuestionByChapter(testPaper.questionList.get(i)
                                                                                .getChapter(),
```

```
                                                                testPaper));
                    } else {
                        testPaper.questionList.set(i, this.questionGenerator.getRandomQuestion());
                    }
                }
            }
        }
    }
}

/***
 * Tekrarlanan soru sayisini verir Geleneksel GA'da ceza amaçlı kullanılacak
 *
 * @param testPaper
 * @param properties
 * @return
 */
public static int duplicationCount(final TestPaper testPaper) {
    final List<String> idList =
testPaper.questionList.stream().map(Question::getId).collect(Collectors.toList());
    return (idList.size() - (int) idList.stream().distinct().count());
}

public static double getPunismentValue(final TestPaper testPaper) {
    final int dupCount = duplicationCount(testPaper);
    final int vioCount = chapterViolationCount(testPaper);
    final double dupFactor = dupCount > 0 ? ((double) dupCount / testPaper.questionList.size()) : 0;
    final double vioFactor = vioCount > 0 ? ((double) vioCount / testPaper.questionList.size()) : 0;
    return dupFactor + vioFactor;
}

public static int randInt(final int min, final int max) {
    final Random rand = new Random();
    final int randomNum = rand.nextInt((max - min) + 1) + min;
    return randomNum;
}

}


package testpaper.generation.repository;

import org.springframework.data.repository.CrudRepository;
import testpaper.generation.model.Question;

public interface QuestionMongoRepository extends CrudRepository<Question, String>{}

package testpaper.generation.repository;

import static org.springframework.data.mongodb.core.query.Criteria.where;
import static org.springframework.data.mongodb.core.query.Query.query;
import static org.springframework.data.mongodb.core.query.Update.update;

import java.util.Collection;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.mongodb.core.MongoTemplate;
```

```
import org.springframework.data.mongodb.core.query.BasicQuery;
import org.springframework.data.mongodb.core.query.Criteria;
import org.springframework.data.mongodb.core.query.Query;
import org.springframework.data.mongodb.core.query.Update;
import org.springframework.stereotype.Repository;

import testpaper.GA.TestPaper;
import testpaper.generation.model.Question;

@Repository
public class QuestionSearchRepository {

   @Autowired
   MongoTemplate mongoTemplate;

   public Collection<Question> searchQuestions(final String text) {
      return this.mongoTemplate.find(Query.query(new
Criteria().orOperator(Criteria.where("text").regex(text, "i"),
                                                Criteria.where("articleTitle").regex(text, "i"),
                                                Criteria.where("chapter").regex(text, "i"))),
                        Question.class);
   }

   public Question getQuestionById(final String id) {
      final BasicQuery query = new BasicQuery("{ questionId : " + id + " }");
      final Question q = this.mongoTemplate.findOne(query, Question.class);
      return q;
   }

   public List<Question> getQuestionsByChapter(final int chapter) {
      final BasicQuery query = new BasicQuery("{chapter: " + chapter + "}");
      final List<Question> qList = this.mongoTemplate.find(query, Question.class);
      return qList;
   }

   public List<Question> getUniqueQuestionsByChapter(final int chapter, final TestPaper testPaper)
{
      final BasicQuery query = new BasicQuery("{chapter: "
                              + chapter
                              + ", questionId:{$nin:["
                              + this.getQuestionIDsByChapter(chapter, testPaper)
                              + "]} }");
      final List<Question> qList = this.mongoTemplate.find(query, Question.class);
      return qList;
   }

   public List<Question> getUniqueQuestions(final TestPaper testPaper) {
      final BasicQuery query = new BasicQuery("{questionId:{$nin:[" +
this.getQuestionIDs(testPaper) + "]} }");
      final List<Question> qList = this.mongoTemplate.find(query, Question.class);
      return qList;
   }

   public void updateSelectionRatio(final Question q) {
      final Query query = query(where("questionId").is(q.getQuestionId()));
      final Update update = update("selectionRatio", q.getSelectionRatio() + 1);
      this.mongoTemplate.updateFirst(query, update, Question.class);
   }

   public void updateSelectionRatio(final TestPaper t) {
```

```
        for (final Question q : t.questionList) {
            this.updateSelectionRatio(q);
        }
    }

    public void resetSelectionRatios() {
        final Query query = query(where("selectionRatio").ne(0));
        final Update update = update("selectionRatio", 0);
        this.mongoTemplate.updateMulti(query, update, Question.class);
    }

    public void resetLastSelections() {
        final Query query = query(where("lastSelection").ne(0));
        final Update update = update("lastSelection", 0);
        this.mongoTemplate.updateMulti(query, update, Question.class);
    }

    public void setRandomSelectionRatio(final int stateSpaceCount) {
        final Random r = new Random();
        for (int i = 1; i <= stateSpaceCount; i++) {
            final Query query = query(where("questionId").is(i));
            final int randomSelectionRatio = r.ints(1, 0, 5).findFirst().getAsInt(); // random value of
(0,1,2,3,4)
            final Update update = update("selectionRatio", randomSelectionRatio);
            this.mongoTemplate.updateFirst(query, update, Question.class);
        }
    }

    public void updateLastSelections(final TestPaper t) {
        for (final Question q : t.questionList) {
            final Query query = query(where("questionId").is(q.getQuestionId()));
            final Update update = update("lastSelection", 1);
            this.mongoTemplate.updateFirst(query, update, Question.class);
        }
    }

    public List<String> getAllChapters() {
        final BasicQuery query = new BasicQuery("{}, {'chapter': 1}");
        List<String> cList = this.mongoTemplate.find(query, String.class);
        cList = cList.stream().distinct().collect(Collectors.toList());
        return cList;
    }

    public String getQuestionIDsByChapter(final int chapter, final TestPaper testPaper) {
        final List<Question> list = testPaper.questionList.stream().filter(q -> q.getChapter() ==
chapter).collect(Collectors.toList());
        String result = "";
        for (final Question question : list) {
            result += question.getQuestionId() + ",";
        }
        if (result.contains(",")) {
            result = result.substring(0, result.lastIndexOf(","));
        }
        return result;
    }

    public String getQuestionIDs(final TestPaper testPaper) {
        String result = "";
        for (final Question question : testPaper.questionList) {
            result += question.getQuestionId() + ",";
```

```
        }
        if (result.contains(",")) {
            result = result.substring(0, result.lastIndexOf(","));
        }
        return result;
    }

}
```

```
package testpaper.generation;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.data.mongodb.repository.config.EnableMongoRepositories;

@SpringBootApplication
@EnableMongoRepositories("testpaper.generation.repository")
public class TestPaperApplication {

    public static void main(final String[] args) {
        SpringApplication.run(TestPaperApplication.class, args);
    }

}
```

```
package testpaper.generation.controllers;

import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import testpaper.generation.model.Question;
import testpaper.generation.repository.QuestionMongoRepository;
import testpaper.generation.repository.QuestionSearchRepository;

@Controller
public class QuestionController {

        @Autowired
        QuestionMongoRepository questionRepository;

        @Autowired
        QuestionSearchRepository questionSearchRepository;

        private List<Question> questionList = new ArrayList<Question>();

        @RequestMapping("/questions")
        public String questions(Model model) {
                model.addAttribute("questionList", questionRepository.findAll());
                return "questions";
        }

        @RequestMapping(value = "/questionSearch")
        public String search(Model model, @RequestParam String search) {
```

```
                model.addAttribute("questionList",
questionSearchRepository.searchQuestions(search));
                model.addAttribute("search", search);
                return "questions";
        }

        public List<Question> getQuestionList() {
                if(questionList.size() == 0){

        questionRepository.findAll().iterator().forEachRemaining(questionList::add);
                }
                return questionList;
        }

}

package testpaper.generation.controllers;

import java.util.Collections;
import java.util.List;
import java.util.Random;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;

import testpaper.GA.Population;
import testpaper.GA.TestPaper;
import testpaper.generation.model.Properties;
import testpaper.generation.model.Question;
import testpaper.generation.repository.QuestionSearchRepository;

@Controller
public class QuestionGenerator {

        @Autowired
        QuestionSearchRepository questionSearchRepository;

        @Autowired
        ConfigController configController;

        /** The random. */
        private static Random random = new Random();

        /***
         * Verilen size bilgisine göre populasyon oluşturur.
         *
         * @param populationSize
         * @param initialise
         */
        public void generatePopulation(Population population) {
                for (int i = 0; i < population.testPapers.length; i++) {
                        TestPaper testPaper = new TestPaper(population.getProperties());
                        this.generateQuestions(testPaper, population.getProperties());
                        population.testPapers[i] = testPaper;
                }
        }

        /**
         * * Yeni random question uretir.
         *
```

```
	 * @return the random question
	 */
	public synchronized Question getRandomQuestion() {
		int randomInt =
random.nextInt(this.configController.properties.getSTATE_SPACE()) + 1;
		final Question question = questionSearchRepository.getQuestionById("" +
randomInt);
		if (question == null) {
			System.out.println("ERROR : question is null for id = " + randomInt);
		}
		return question;
	}

	/**
	 * * Verilen test sayfasinda olmayan yeni bir random question uretir.
	 *
	 * @return the random question
	 */
	public synchronized Question getUniqueRandomQuestion(TestPaper testPaper) {
		final List<Question> questions =
questionSearchRepository.getUniqueQuestions(testPaper);
		int randomInt = random.nextInt(questions.size());
		Question question = questions.get(randomInt);
		if (question == null) {
			System.out.println("ERROR : question is null for id = " + randomInt);
		}
		return question;
	}

	/**
	 * * Verilen test sayfasina ve istenilen chapter'a göre yeni bir random question uretir.
	 *
	 * @return the random question
	 */
	public synchronized Question getRandomUniqueQuestionByChapter(int chapter, TestPaper
testPaper) {
		final List<Question> questions =
questionSearchRepository.getUniqueQuestionsByChapter(chapter, testPaper);
		int randomInt = random.nextInt(questions.size());
		Question question = questions.get(randomInt);
		return question;
	}

	/**
	 * * Chapter'a göre istenilen sayıda random question uretir. Popülasyonun
	 * ilk oluşturulmasında chapter bilgisine göre soruları random üretmek için
	 * kullanılır.
	 *
	 * @return the random question
	 */
	public synchronized void getRandomQuestionsByChapter(int chapter, int amount,
TestPaper testPaper) {
		if (amount > 0) {
			final List<Question> randomQuestions =
questionSearchRepository.getQuestionsByChapter(chapter);
			// shuffle list
			Collections.shuffle(randomQuestions);
			// adding numbers to random list
			for (int j = 0; j < amount; j++) {
				testPaper.questionList.add(randomQuestions.get(j));
```

```
                        }
                    }
                }

}


package testpaper.generation.controllers;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import testpaper.GA.FitnessCalc;
import testpaper.GA.Population;
import testpaper.generation.model.Properties;
import testpaper.generation.model.Question;
import testpaper.generation.repository.QuestionSearchRepository;

@Controller
public class GAController {

    @Autowired
    QuestionSearchRepository questionSearchRepository;

    @Autowired
    QuestionGenerator       questionGenerator;

    @Autowired
    ConfigController        configController;

    @Autowired
    Algorithm               algorithm;

    public String           result;
    public String           resultQuestionIDs;
    public String           resultQuestionsDificulty;
    public String           resultIteration;
    public String           resultFitness;

    public Population       population;
    public List<Question>   resultQuestions;

    /***
     * Constructor.
     */
    public GAController() {
    }

    @RequestMapping(value = "/startGA", method = RequestMethod.GET)
    public String test(final Model model) {
        model.addAttribute("DEFAULT_GENE_LENGTH",
this.configController.properties.getDEFAULT_GENE_LENGTH());
        model.addAttribute("POPULATION_SIZE",
this.configController.properties.getPOPULATION_SIZE());
```

```
        model.addAttribute("MAX_ITERATION",
this.configController.properties.getMAX_ITERATION());
        model.addAttribute("DIFFICULTY", this.configController.properties.getDIFFICULTY());
        model.addAttribute("SCORE", this.configController.properties.getSCORE());
        model.addAttribute("KNOWLEDGE_POINT",
this.configController.properties.getKNOWLEDGE_POINT());
        model.addAttribute("ESTIMATED_ANSWER_TIME",
this.configController.properties.getESTIMATED_ANSWER_TIME());
        model.addAttribute("chapter1", this.configController.properties.getChapter1());
        model.addAttribute("chapter2", this.configController.properties.getChapter2());
        model.addAttribute("chapter3", this.configController.properties.getChapter3());
        model.addAttribute("chapter4", this.configController.properties.getChapter4());
        model.addAttribute("chapter5", this.configController.properties.getChapter5());
        model.addAttribute("w1", this.configController.properties.getW1());
        model.addAttribute("w2", this.configController.properties.getW2());
        model.addAttribute("w3", this.configController.properties.getW3());
        model.addAttribute("w4", this.configController.properties.getW4());
        model.addAttribute("excludeLastQuestions",
this.configController.properties.isExcludeLastQuestions());
        return "startGA";
    }

    @RequestMapping(value = "/startGA", method = RequestMethod.POST)
    public String constant(final Model model, @ModelAttribute
    final Properties constant) {

this.configController.properties.setDEFAULT_GENE_LENGTH(constant.getDEFAULT_GENE_LE
NGTH());
        this.configController.properties.setChapter1(constant.getChapter1());
        this.configController.properties.setChapter2(constant.getChapter2());
        this.configController.properties.setChapter3(constant.getChapter3());
        this.configController.properties.setChapter4(constant.getChapter4());
        this.configController.properties.setChapter5(constant.getChapter5());
        this.configController.properties.setW1(constant.getW1());
        this.configController.properties.setW2(constant.getW2());
        this.configController.properties.setW3(constant.getW3());
        this.configController.properties.setExcludeLastQuestions(constant.isExcludeLastQuestions());

        model.addAttribute("DEFAULT_GENE_LENGTH",
this.configController.properties.getDEFAULT_GENE_LENGTH());
        model.addAttribute("POPULATION_SIZE",
this.configController.properties.getPOPULATION_SIZE());
        model.addAttribute("MAX_ITERATION",
this.configController.properties.getMAX_ITERATION());
        model.addAttribute("DIFFICULTY", this.configController.properties.getDIFFICULTY());
        model.addAttribute("SCORE", this.configController.properties.getSCORE());
        model.addAttribute("KNOWLEDGE_POINT",
this.configController.properties.getKNOWLEDGE_POINT());
        model.addAttribute("ESTIMATED_ANSWER_TIME",
this.configController.properties.getESTIMATED_ANSWER_TIME());
        model.addAttribute("chapter1", this.configController.properties.getChapter1());
        model.addAttribute("chapter2", this.configController.properties.getChapter2());
        model.addAttribute("chapter3", this.configController.properties.getChapter3());
        model.addAttribute("chapter4", this.configController.properties.getChapter4());
        model.addAttribute("chapter5", this.configController.properties.getChapter5());
        model.addAttribute("w1", this.configController.properties.getW1());
        model.addAttribute("w2", this.configController.properties.getW2());
        model.addAttribute("w3", this.configController.properties.getW3());
        model.addAttribute("w4", this.configController.properties.getW4());
```

```
        model.addAttribute("excludeLastQuestions",
this.configController.properties.isExcludeLastQuestions());

        this.start();

        model.addAttribute("hdnLabels", this.getLabels());
        model.addAttribute("hdnFitness", this.resultFitness);
        model.addAttribute("resultQuestions", this.population.getFittest().getQuestionList());

        return "result";
    }

    /***
     * Algoritmayı baslatmak icindir
     */
    public void start() {

        // Ekrandan girilen değerler

FitnessCalc.setDesiredQuestionCount(this.configController.properties.getDEFAULT_GENE_LENG
TH());
        FitnessCalc.setDesiredDifficulty(this.configController.properties.getDIFFICULTY());
        FitnessCalc.setDesiredScore(this.configController.properties.getSCORE());

FitnessCalc.setDesiredAnswerTime(this.configController.properties.getESTIMATED_ANSWER_TI
ME());

FitnessCalc.setDesiredKnowledgePoint(this.configController.properties.getKNOWLEDGE_POINT()
);

        // Populasyonu yapılandırır
        this.population = new Population(this.configController.getProperties());
        this.questionGenerator.generatePopulation(this.population);

        int generationCount = 0;

        // Amaç fonksiyonunu ve fitness değerlerini hesapla
        this.population.targetFunction();
        this.algorithm.sortAndCalculateFitness(this.population);
        // FITNESS'a göre sıralama

        final StringBuilder sb = new StringBuilder();

        // f(x)'e göre değişiyor ***
        while ((this.population.getFittest().getFitnessValue() <= 10) ||
(this.population.getFittest().getPunishment() != 0)) {
            generationCount++;
            System.out.println("Iteration: " + generationCount + " Fittest: " +
this.population.getFittest().toDetailString());

            this.population = this.algorithm.evolvePopulation(this.population,
this.configController.getProperties());

            this.population.targetFunction();
            this.algorithm.sortAndCalculateFitness(this.population);

            // tekrarli soru olma ihtimaline karsi
            this.algorithm.punishmentControl(this.population.getFittest(),
this.population.getProperties());
```

```
            sb.append(this.population.getFittest().getFitnessValue() + "-");

            if (generationCount >= this.configController.properties.getMAX_ITERATION()) {
                System.out.println("THE LAST ITERATION COMPLETED !!!");
                break;
            }
        }

        this.resultFitness = sb.toString();
        this.result = this.population.getFittest().toResultString();
        this.resultIteration = "" + generationCount;
        this.population.getFittest().sortByQuestionID();
        this.resultQuestionIDs = this.population.getFittest().getQuestionsAsString();
        this.resultQuestionsDificulty = this.population.getFittest().getDifficultiesAsString();

        System.out.println(this.population.getFittest().toDetailString());
        System.out.println("Iteration    : " + generationCount + " Questions: " +
this.population.getFittest().getQuestionListAsString());
        System.out.println("Question IDs : " + this.resultQuestionIDs);
        System.out.println("Difficulties : " + this.resultQuestionsDificulty);
        System.out.println("Know. Points : " +
this.population.getFittest().getKnowledgePointsAsString());
        System.out.println("Q.    Scores : " + this.population.getFittest().getScoresAsString());
        System.out.println("Q.   S.Ratio : " + this.population.getFittest().getSelectionRatioAsString());
        System.out.println("Q.  Chapters : " +
this.population.getFittest().getQuestionChaptersAsString());
        System.out.println("Answer Times : " +
this.population.getFittest().getQuestionAnswerTimesAsString());
    }

    private String getLabels() {
        final StringBuilder sb = new StringBuilder();
        for (int i = 1; i <= this.configController.properties.getMAX_ITERATION(); i++) {
            sb.append(i);
            sb.append("-");
        }
        return sb.toString();
    }

    @RequestMapping(value = "/result", method = RequestMethod.GET)
    public String result(final Model model, @ModelAttribute
    final Properties constant) {

this.configController.properties.setDEFAULT_GENE_LENGTH(constant.getDEFAULT_GENE_LE
NGTH());
        this.configController.properties.setChapter1(constant.getChapter1());
        this.configController.properties.setChapter2(constant.getChapter2());
        this.configController.properties.setChapter3(constant.getChapter3());
        this.configController.properties.setChapter4(constant.getChapter4());
        this.configController.properties.setChapter5(constant.getChapter5());
        this.configController.properties.setW1(constant.getW1());
        this.configController.properties.setW2(constant.getW2());
        this.configController.properties.setW3(constant.getW3());
        this.configController.properties.setExcludeLastQuestions(constant.isExcludeLastQuestions());

        model.addAttribute("DEFAULT_GENE_LENGTH",
this.configController.properties.getDEFAULT_GENE_LENGTH());
        model.addAttribute("POPULATION_SIZE",
this.configController.properties.getPOPULATION_SIZE());
```

```
        model.addAttribute("MAX_ITERATION",
this.configController.properties.getMAX_ITERATION());
        model.addAttribute("DIFFICULTY", this.configController.properties.getDIFFICULTY());
        model.addAttribute("SCORE", this.configController.properties.getSCORE());
        model.addAttribute("KNOWLEDGE_POINT",
this.configController.properties.getKNOWLEDGE_POINT());
        model.addAttribute("ESTIMATED_ANSWER_TIME",
this.configController.properties.getESTIMATED_ANSWER_TIME());
        model.addAttribute("chapter1", this.configController.properties.getChapter1());
        model.addAttribute("chapter2", this.configController.properties.getChapter2());
        model.addAttribute("chapter3", this.configController.properties.getChapter3());
        model.addAttribute("chapter4", this.configController.properties.getChapter4());
        model.addAttribute("chapter5", this.configController.properties.getChapter5());
        model.addAttribute("w1", this.configController.properties.getW1());
        model.addAttribute("w2", this.configController.properties.getW2());
        model.addAttribute("w3", this.configController.properties.getW3());
        model.addAttribute("w4", this.configController.properties.getW4());
        model.addAttribute("excludeLastQuestions",
this.configController.properties.isExcludeLastQuestions());
        model.addAttribute("hdnLabels", this.getLabels());
        model.addAttribute("hdnFitness", this.resultFitness);
        model.addAttribute("resultQuestions", this.population.getFittest().getQuestionList());

        return "result";
    }

    public Population getPopulation() {
        return this.population;
    }

    public void setPopulation(final Population population) {
        this.population = population;
    }

    public List<Question> getResultQuestions() {
        return this.resultQuestions;
    }

    public void setResultQuestions(final List<Question> resultQuestions) {
        this.resultQuestions = resultQuestions;
    }

}
```

ÖZGEÇMİŞ

Ad Soyad: Ufuk TÜL

Doğum Yeri ve Tarihi: Elazığ / 1980

E-Posta: ufuktul@gmail.com

Lisans: Fırat Üni. Bilgisayar Mühendisliği 2003

Mesleki Deneyim ve İlgi Alanları: Open source platform çözümleri, Java teknolojileri, makine öğrenmesi, yapay sinir ağları, derin öğrenme, yapay zeka ve sezgisel algoritmalar

TEZDEN TÜRETİLEN YAYINLAR

▪ **Tül, U., Tuncer, A.,** 2017: Genetik Algoritma ile Akıllı Test Sayfası Oluşturma. Gazi Üniversitesi Fen Bilimleri Dergisi Part C: Tasarım ve Teknoloji, 5 (4), 27-34. DOI: 10.29109/http-gujsc-gazi-edu-tr.341977