YAZILIM PROJELERİNİN OPTİMİZASYON PROBLEMİ OLARAK İNCELENMESİ VE GENETİK ALGORİTMA İLE ÇÖZÜMÜ

Yücel Dil

Danışman
Yrd. Doç. Dr. Mustafa Cem Kasapbaşı

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
İSTANBUL - 2015

KABUL VE ONAY SAYFASI

Yücel DİL tarafından hazırlanan **"Yazılım Projelerinin Optimizasyon Problemi Olarak İncelenmesi ve Genetik Algoritma ile Çözümü "** adlı tez çalışması 29/01/2016 tarihinde, aşağıdaki jüri üyeleri önünde başarı ile savunularak, İstanbul Ticaret Üniversitesi Fen Bilimleri Enstitüsü **Bilgisayar Mühendisliği Anabilim Dalı**'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Danışman	**Yrd. Doç. Dr. Mustafa Cem Kasapbaşı** İstanbul Ticaret Üniversitesi	
Jüri Üyesi	**Doç. Dr. Serhat Özekes** Üsküdar Üniversitesi	
Jüri Üyesi	**Yrd. Doç. Dr. Alper Özpınar** İstanbul Ticaret Üniversitesi	

Onay Tarihi : .../.../....

Prof. Dr. Doğan KAYA
Enstitü Müdürü

AKADEMİK VE ETİK KURALLARA UYGUNLUK BEYANI

İstanbul Ticaret Üniversitesi, Fen Bilimleri Enstitüsü, tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmasında,

- tez içindeki bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- görsel, işitsel ve yazılı tüm bilgi ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu,
- başkalarının eserlerinden yararlanılması durumunda ilgili eserlere bilimsel normlara uygun olarak atıfta bulunduğumu,
- atıfta bulunduğum eserlerin tümünü kaynak olarak gösterdiğimi,
- kullanılan verilerde herhangi bir tahrifat yapmadığımı,
- ve bu tezin herhangi bir bölümünü bu üniversitede veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.

Tarih 29.01.2016

İmza

Yücel Dil

İÇİNDEKİLER

Sayfa

ÖZET

Yüksek Lisans Tezi

YAZILIM PROJELERİNİN OPTİMİZASYON PROBLEMİ OLARAK İNCELENMESİ VE GENETİK ALGORİTMA İLE ÇÖZÜMÜ

Yücel Dil

İstanbul Ticaret Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Yrd. Doç. Dr. Mustafa Cem Kasapbaşı

2015, 82 sayfa

Optimizasyon problemlerinin çözümü için kullanılan bir çok optimizasyon tekniği, doğadaki olaylardan esinlenilerek geliştirilmiştir. Genetik algoritmalar(GA), canlıların doğada geçirdikleri süreci örnek alan ve çevre şartlarına uyum sağlayan nesillerin yaşamlarını koruması, bu uyumu sağlayamayan nesillerin yok olması temelinde ele alınan bir optimizasyon algoritmasıdır.

Bu çalışmada Genetik algoritma tanımı ve çalışma prensibi ele alınmış, matlab programlama dili kullanılarak uygun sonuçların elde edilmesi sağlanmıştır. Genetik algoritmaların optimizasyon problemlerinde kullanımı için doğrusal ve doğrusal olmayan optimizasyon problemleri incelenmiş ve genetik algoritma ile çözümleri gerçekleştirilmiştir. Kurgulanan optimizasyon problemleri için yazılım proje yönetimi konusu seçilmiştir. Bu bağlamda yazılım projeleri ve yönetimi konusu, çalışmada detaylı olarak incelenmiştir.

Anahtar Kelimeler: Genetik algoritma, Optimizasyon, Yazılım projesi, Proje yönetimi.

ABSTRACT

M.Sc. Thesis

AN OPTIMIZATION MODEL AND GENETIC ALGORITHM SOLUTION FOR SOFTWARE PROJECTS

Yücel Dil

İstanbul Commerce University
Graduate School of Applied and Natural Sciences
Department of Computer Engineering

Supervisor: Assist. Prof. Dr. Mustafa Cem Kasapbaşı

2015, 82 pages

Many optimization techniques have been developed by inspired from the events of nature for the solution of optimization problems. Genetic algorithms which are based on imitation of the period spent in the nature of the live specimens space and protection of the lives of generations to adapt to environmental conditions and extinction of those which cannot adapt to this situation.

The description of the genetic algorithm and about how it works were examined also using matlab program for proper result is proposed in this study. Linear and nonlinear problems were examined for optimization problems were carried out with using genetic algorithm. The topic of software project managment has been chosen for edited optimization problems. Accordingly, the topic of software project and managment were detail examined in this study.

Keywords: Genetic algorithm, Optimization, Software project, Project management

TEŞEKKÜR

Bu araştırma için beni yönlendiren, karşılaştığım zorlukları bilgi ve tecrübesi ile aşmamda yardımcı olan değerli Danışman Hocam Yrd. Doç. Dr. Mustafa Cem Kasapbaşı'na teşekkürlerimi sunarım.

Tezimin imalat aşamasındaki desteklerinden dolayı çalışma arkadaşlarım ve saygıdeğer hocalarıma teşekkür ederim.

Tezimin her aşamasında beni yalnız bırakmayan sevgili eşim Sema Özenç Dil'e ve başımın tacı çok değerli annem ve babama sonsuz sevgi ve saygılarımı sunarım.

Yücel Dil
İSTANBUL, 2015

ŞEKİLLER

ÇİZELGELER

SİMGELER VE KISALTMALAR

A_n	Analist
C	Maliyet
EP	Evrimsel programlama
ES	Evrimsel strateji
GA	Genetik algoritma
GP	Genetik programlama
l	Kromozomdaki gen sayısı
N	Populasyondaki kromozom sayısı
p_c	Çaprazlama operatörü uygulama olasılığı
p_m	Mutasyon olasılığı
X_n	Fonksiyon değişkeni
Y_n	Yazılım geliştirici

1. GİRİŞ

Mühendislik problemleri genellikle bir amaca yönelik en iyi çözüm bulacak optimizasyon problemleri şeklinde ifade edilirler. Bu problemler doğrusal veya doğrusal olmayan şekilde tanımlanabilir. Bu problemlerin çözümü için optimizasyon yöntemleri kullanılır. Bu nedenle optimizasyon yöntemleri, mühendislik çalışmaları açısından önemli bir konudur. Bilgisayarların yaygınlaşması ile birlikte bu çözümler daha kolay ve kesinleşmiş, ancak karmaşık ve büyük problemlerin çözümünde bilgisayarların da yetersiz kaldığı durumlar ortaya çıkmıştır. Bunun nedeni çoğu kez problemlerin analitik çözümlerinin olmaması ve bilgisayarlarla yapılan iterasyona dayalı çözüm yöntemlerinde yaşanan hız sorunlarıdır. Bu sorunlar yeni çözüm yöntemlerinin geliştirilmesine ortam hazırlamıştır.

Bu bağlamda araştırmalar doğadaki olaylardan esinlenilerek birçok optimizasyon tekniğinin geliştirilmesine olanak sağlamıştır. Genetik algoritmalar da doğadan esinlenilerek geliştirilmiştir.

Genetik algoritmalar canlıların doğada geçirdikleri süreci örnek alan ve çevre şartlarına uyum sağlayan nesillerin yaşamlarını koruması, bu uyumu sağlayamayan nesillerin yok olması temelinde ele alınan bir optimizasyon algoritmasıdır. İlk olarak, Michigan Üniversitesi'nden John Holland tarafından 1975 yılında geliştirilmiştir. Holland (1975), yaptığı araştırmalara, doğal sistemlerin süreçlerini açıklamak ve doğal sistemlerin işleme yordamını içeren yapay sistem yazılımları tasarlamak amacı ile başlamıştır. Bu yaklaşım hem doğal hem de yapay sistemlerde önemli yeniliklerin yapılmasına neden olmuştur.

Bu çalışmada genetik algoritmaların optimizasyon problemlerinde kullanımı için doğrusal ve doğrusal olmayan optimizasyon problemleri incelenmiş ve genetik algoritma ile çözümleri gerçekleştirilmiştir. Kurgulanan optimizasyon problemleri için yazılım proje yönetimi konusu seçilmiştir.

Yazılım proje yönetimi konusu özellikle son yıllarda önem kazanmıştır. Yazılımların hizmet verdiği alanların artması ve kritik hale gelmesi, yazılımdan beklenen kalite beklentilerini yükseltmiştir. Yazılım geliştirme süreç analizinden başlayan ve kodlama, test ve canlı kullanıma almaya kadar uzanan bir süreçtir. İyi planlanması ve yönetilmesi gerekir. Kullanıcı isteklerinin kaliteli, zamanında ve bütçesinde karşılanması, elbette analiz, geliştirme ve test süreçlerinin kalitesi ile proje yönetiminin doğru yapılmasına bağlıdır. Bu çalışma neticesinde, yazılım proje yönetimi konusuna, farklı bir bakış ile katkı sağlandığı düşünülmektedir.

2. LİTERATÜR ÖZETİ

Genetik algoritmalar ilk olarak Michigan Üniversitesi'nden John Holland tarafından doğal seçim ve genetik popülasyonların modellenmesi olarak 1975 yılında geliştirilmiştir. Holland (1975), evrimden ve canlılardaki bu süreçten yararlanarak, makine öğrenmesi üzerine çalışmalarını yoğunlaştırmıştır. Bu süreci bilgisayar ortamında modelleyerek tek bir mekanik yapının öğrenme yeteneğini geliştirmek yerine böyle yapılardan oluşan topluluğun çiftleşme, çoğalma ve mutasyon gibi genetik süreçlerden geçerek başarılı yeni bireyler oluşturabildiğini görmüştür.

Holland (1975), genetik algoritmayı doğrusal olmayan çok değişkenli optimizasyon problemlerinin çözümünde kullanarak önemli bir araştırma algoritması olduğunu kanıtlamıştır. Genetik algoritmanın kısıtlı optimizasyon problemleri kapsamına giren çizelgeleme yöntemlerinden biri olan atölye çizelgeleme problemlerinde etkili olarak ilk defa Davis tarafından 1985 yılında kullanılmıştır (Davis, 1985).

Goldberg 1989 yılında, genetik algoritmayı rastlantısal arama tekniklerini kullanarak çözüm bulmaya çalışan, parametre kodlama esasına dayanan sezgisel bir arama tekniği olarak tanımlamıştır. Goldberg bu çalışma ile genetik algoritmanın dünyanın her yerinde farklı konularda kullanılabileceğini göstermiştir (Goldberg, 1989).

Başlangıçta sürekli olmayan en iyileme problemlerine uygulanan genetik algoritmalar, sonraları gezgin satıcı, karesel atama, yerleşim, atölye çizelgeleme, ders/sınav programı hazırlanması gibi problemlerde başarıyla uygulanmıştır. Son yıllarda üretim planlama, elektronik ve finansman gibi farklı ve geniş alanlara yönelik gerek teorik gerekse uygulamalı genetik algoritma çalışmalarının sayısı artmaktadır.

Genetik algoritmalar, sezgisel bir yöntem olduğundan dolayı verilen bir problem için kesin sonucu bulamayabilir. Ancak bilinen yöntemlerle çözülemeyen ya da

çözüm zamanı problemin büyüklüğü ile üssel olarak artan problemlerde kesin sonuca yakın çözümler verdiğinden kullanılabilir. Genetik algoritmaların çizelgeleme, makine öğrenmesi, tasarım, hücresel üretim gibi alanlarda başarılı uygulama alanları bulunmaktadır (Biegal ve Davern, 1990).

Karmaşık problemleri hızlı ve optimale yakın olarak çözebilen genetik algoritmalar bunun dışında , parametrik bir yaklaşımla çözüm aradığı için çok geniş aralıkta problem çeşidine uygulanabilir. Genetik algoritmaların rastsal arama yöntemleri olmalarına rağmen klasik rastsal arama yöntemlerinden bazı farklılıkları bulunmaktadır. Bu farklılıklar; değişkenlerle değil, değişken kümesinin kodlanmış biçimi ile çalışmalarından, tek bir noktadan değil, noktalar kümesi üzerinden arama yapmalarından, türev ve benzeri yardımcı bilgi kullanmamalarından ve yalnızca amaç fonksiyonu bilgisine ihtiyaç duymalarından kaynaklanmaktadır (Goldberg, 1989).

Yönetim kavramı tarih içinde birçok farklı aşama geçirmiştir. Tarih boyunca hemen herkes yönetici veya yönetilen pozisyonunda bulunmuştur. Sanayi devriminden önce kapalı toplumlarda daha çok otoriteye dayalı yönetim tarzı uygulanmıştır. Kişiler, dini ve monarşiye dayalı otorite veya toprağa bağlı zenginlikleriyle toplumu yönetmiştir. Bu dönemde yönetim bir bilim olarak olgunlaşmamıştır (Drucker, 2006).

Sanayi devrimi, üretimi artırmaya ve otomasyona yönelik yönetim anlayışıyla insani değerleri geri plana itse de; süreci, kullanılan tekniği ve sistemi öne çıkartan bir yönetim tarzı oluşturmuştur. 19 ve 20. yüzyıldaki ekonomik ve sosyal buhranlar ise insan faktörünün yeniden gündeme gelmesini sağlamıştır.

Proje Yönetimi konusundaki çalışmalar ise 1950'li yıllardan itibaren ivme kazanmıştır. Özellikle büyük ölçekli projelerin artması, projelerin en iyi şekilde yönetilmesi gerekliliğini ortaya çıkarmış ve CPM (Critical Path Method – Kritik Yol Yöntemi) ve PERT (Project Evaluation and Review Technique – Program Değerlendirme ve Gözden Geçirme Yöntemi) gibi ağ tabanlı teknikler 1950'lerin sonunda ortaya çıkmıştır. Bu gelişmeleri takiben konuya ilgi artmış ve çok sayıda

çalışma yapılmıştır. Belirsizliklerin yüksek olduğu sistemlerin yönetimi, 1990'lı yıllarda Proje yönetimine ilişkin farklı bir bakış kazandırmış ve çizelgeleme yöntemi proje yönetimi için kullanılmıştır. Literatürde RCPSP (The Resource Constrained Project Scheduling - Kaynak Kısıtlı Proje Çizelgeleme) olarak geçen çizelgeleme yöntemi ile özellikle zaman odaklı amaçları olan problemlerin çözümü sağlanmıştır.

Günümüz artık bilgi çağı olarak isimlendiriliyor. Yönetim konusunun tam bir bilim olarak ele alınması da bu çağda hızlanmıştır. Yapılan çalışmalara ve araştırmalara rağmen yönetim konusunda hala birçok belirsizlik olduğu söylenebilir. Bilgi çağında ortaya çıkan yazılım geliştirme ve yönetimi konusunda yapılan çalışmalar ise henüz yeni olarak kabul edilebilir.

Yazılım, hayatın her alanında kişi ve kurumların çok çeşitli ihtiyaçlarına çözüm sunmaktadır. Teknolojideki gelişmeler, kurumlar arası rekabet, müşteri beklentilerindeki artış yazılım ve yazılım geliştirme çalışmalarının önemini artırmaktadır. Bu öneme cevap verebilecek çalışmalar gerçekleştirmenin yolu yapılacakların kapsamını belirlemek, planlamak, ekipler oluşturmak, plana bağlı çalışmak ve işleri belirlenen sürede tamamlamaktır. Bu çalışmaların tümü yazılım projesi olarak adlandırılır (Albayrak, 2005).

Yazılım ürünü ve geliştirme süreci diğer mühendislik dallarına göre oldukça farklıdır. Öncelikle yazılım mühendisliği diğer bilimlere göre daha yeni bir alandır ve yaklaşık 60 yıllık bir maziye sahiptir. Bu kısa geçmiş, yazılım alanında herkes tarafından kabul edilen standart çözümlerin sayısını düşürmektedir. Bu da yöntem belirlemeyi ve standartlaşmaya gitmeyi zorlaştırmıştır. Sürekli değişen ihtiyaç ve teknolojiler de bu duruma katkı sağlamaktadır. Yazılım ürününü ya da bir bütün olarak ele alınırsa projesini diğer proje yaklaşımlarından ayıran özellikler karmaşıklık, uyumluluk, değişkenlik ve soyut olması şeklinde ifade edilmiştir (Brooks, 1995).

Yazılım projelerinin hedeflediği alan da diğer mühendislik dallarından farklıdır. Diğer mühendislik dalları makine ve cihaz tasarımı gibi hayatı kolaylaştıran

ürünlerle ilgilenirken, birçok yazılım ürünü doğrudan hayatın kendisini modellemeye yönelmektedir. Bu hedef, hayatın tüm karmaşasını yazılım ürününe dolayısıyla yazılım projesine taşır. Kişisel bakış açısı ve insani ilişkiler daha ön plana çıkar ve teknik olarak doğru yapılan tüm çalışmalar başarı için yeterli olmayabilir (Ören vd., 2006).

3. EVRİMSEL ALGORİTMALAR

Evrim, çevre şartlarına en iyi uyum sağlayan bireylerin hayatta kalarak, nesilden nesile bu kalıtsal değişikliklerini aktarması olarak ifade edilebilir. Evrimsel algoritmalar biyolojinin bu temel yaklaşımından esinlenen hesaplama yöntemleridir. Evrimsel algoritmaları mühendislik alanlarında uygulamak mümkündür. Bilgisayar bilimleri için düşünürsek, özellikle yapay zeka çalışmalarında faydalanılan bir konudur.

Evrimsel algoritmalar için genel işleyiş Çizelge 3.1 'de kısaca ifade edilmektedir.

Çizelge 3.1 Evrimsel Algoritmanın Genel işleyişi (Dipankar ve Zbigniew, 1997)

```
Başla
        t = 0
        İlk çözüm uzayı P(t)'yi oluştur
        İlk çözüm uzayı P(t)'nin uyum değerini hesapla
                Durdurma kriteri sağlanana kadar
                Başla
                        t = t + 1
                        P(t-1)'den P(t)'yi seç
                        P(t)'yi operatörler ile değişime
uğrat
                        P(t)'nin uyum değerini hesapla
        Dur
```

Evrimsel algoritma her t anında bireylerden oluşan bir P(t) popülasyonunu içerir. Her bir birey, söz konusu problem için bir adet potansiyel çözümü ifade eder. Her çözümün uyum değeri hesaplanır. Bir sonraki adımda uyum değeri yüksek bireyler seçilerek yeni bir popülasyon oluşturulur. Bu yeni popülasyonun bazı üyeleri operatörler yardımıyla değişime uğratılır. Bu operatörlerin en bilinenleri çaprazlama ve mutasyondur. Yeni popülasyonun uyum değeri hesaplanır ve tüm bu adımlar bir durdurma kriteri sağlanana kadar devam eder.

Genel olarak evrimsel algoritmalar aşağıdaki gibi sınıflandırılabilir;

Genetik algoritmalar (GA),
Genetik programlama (GP),
Evrimsel programlama (EP)
Evrimsel strateji (ES).

Bu çalışmada Genetik algoritma konusu işlenecektir. Genetik algoritma ve genetik programlama genetik seviyede evrimin modelleridir. Evrim stratejisinde ise popülasyondaki bireylerin yapıları optimize edilmeye çalışılır. Bireylerin çeşitli davranışsal özellikleri parametrik hale getirilir ve optimizasyon süresinde bu değerler gelişir. Evrimsel programlamada ise çeşitli türlerin davranışsal özelliklerinin adaptasyonu üzerinde durularak en üst düzey soyutlama kullanılır. (Wong ve Leung, 1999)

4. GENETİK ALGORİTMALAR

4.1. Giriş

Genetik algoritmalar, doğadaki canlıların geçirdiği süreci örnek alır ve iyi nesillerin kendi yaşamlarını korurken, kötü nesillerin yok olması ilkesine dayanır. Darwin'in çevre şartlarına uyum sağlayabilen en iyilerin hayatta kalması ilkesinden hareket eder. Matematiksel modellemenin yapılamadığı veya kesin çözümün olmadığı problemlerde genetik algoritmadan yararlanılır. Bu algoritma, anne ve baba bireyden (bir önceki nesil) doğan yeni bireylerin şartlara uyum sağlayıp yaşamlarını devam ettirmesine dayanır. Yeni bireyler, ebeveynlerinden gelen iyi genleri bünyelerinde muhafaza edebileceği gibi kötü genleri de almış olabilir. Böyle bir durumda kötü genlere sahip bireyler elenecektir.

Genetik algoritma, rastlantısal arama tekniklerini kullanarak çözüm bulmaya çalışan, parametre kodlama esasına dayanan sezgisel bir arama tekniğidir. (Goldberg, 1989)

4.2. Tarihçe

Genetik algoritma evrimsel algoritmaların rastlantısal araştırma yöntemlerini kullanarak kendi kendine öğrenme ve karar verme sistemlerinin düzenlenmesine yönelik bir araştırma tekniğidir. İlk olarak, Michigan Üniversitesi'nden John Holland tarafından 1975 yılında geliştirilmiştir. John Holland'ın araştırmaları, doğal sistemlerin süreçlerini açıklamak ve doğal sistemlerin işleme yordamını içeren yapay sistem yazılımları tasarlamak amacı ile başlamıştır. Bu yaklaşım hem doğal hem de yapay sistemlerde önemli yeniliklerin yapılmasına neden olmuştur. Bu çalışmadan sonra genetik algoritmalar dünyanın her yerinde farklı konularda kullanılmıştır.

Holland (1975), çalışmalarında bir sistemin girişinde genetik bileşen kullanarak örnek bir makine öğrenme tekniği geliştirmiştir. Bu çalışmadan sonra genetik algoritma doğrusal olmayan çok değişkenli eniyileme problemlerinin çözümünde

kullanılarak önemli bir araştırma algoritması olduğu kanıtlanmıştır. Genetik algoritmanın kısıtlı optimizasyon problemlerinde kullanılması ilk olarak Davis tarafından çizelgeleme yöntemlerinden biri olan atölye çizelgeleme problemlerinde 1985 yılında kullanılmıştır.

4.3. Temel Kavramlar

Genetik algoritmalar doğal evrim sürecine benzer olduğu için, genetik algoritmalarda kullanılan terminoloji de, evrim kuramı ile benzerlik taşır. Doğada, bir popülasyondaki bireyler yiyecek, çiftleşme ve barınma için diğer bireylerle rekabet halindedir. Rakiplerine oranla çevreye daha iyi uyum sağlayan bireyler, çevreye daha az uyum sağlayan diğer bireylere göre fazla hayatta kalır ve çoğalırlar. Bu bireylerin özellikleri genlerine kodlanarak yavrularına geçer ve gelecek kuşaklarda uyumu daha az olan bireylere göre daha fazla temsil edilirler. Bu şekilde gen vasıtasıyla bilgiler bir sonraki kuşağa aktarılmaktadır. (Yeniay, 2001)

Evrimsel kuramda yer alan terimlerin genetik algoritmalardaki karşılığı Çizelge 4.1.'de verilmiştir.

Çizelge 4.1. Evrimsel kuram ve Genetik Algoritma Kavramlarının karşılaştırılması (Goldberg, 1989)

Evrimsel Kuram	**Genetik Algoritma**
Gen	Nitelik
Kromozom	Dizi
Allel	Nitelik değeri
Lokus	Dizi konumu
Genotip	Yapı
Fenotip	Parametre kümesi, alternatif çözüm
Epistasis	Doğrusal olmama

4.3.1. Gen

Kromozom yapısındaki kendi başına birer genetik bilgi taşıyan en ufak yapı birimine gen denir. Kısmi bilgiler taşıyan bu ufak yapıların bir araya gelmesiyle bütün bir çözüm kümesini oluşturan kromozom(dizi) meydana gelir.

Genetik algoritmanın kullandığı programlama yapısında bu gen yapıları programcının tanımlamasına bağlıdır. Bir genin içerdiği bilgi sadece ikili tabandaki sayıları içerebildiği gibi onluk taban ve onaltılık tabandaki sayı değerlerini de içerebilir. Dolayısıyla yazılan programa göre gen içeriği çok önem kazanmaktadır. (Elmas, 2011)

4.3.2. Kromozom

Bir ya da birden fazla gen yapısının bir araya gelerek problemin çözümüne ait tüm bilgiyi içeren dizilere kromozom denir. Kromozomların bir araya gelmesiyle popülasyon (yığın) oluşturulur. Yığındaki her bir bireye kromozom, kromozomdaki her bir bilgiye gen denir. Kromozomlar, üzerinde durulan problemin olası çözüm bilgilerini içermektedir.

Kromozomlar, genetik algoritma yaklaşımında üzerinde durulan en önemli birim olduğu için bilgisayar ortamında iyi ifade edilmeleri gerekmektedir. Kromozomun hangi kısmının ne anlam taşıyacağı, ne tür bilgi içereceği kullanıcının olaya bakışını değiştirmektedir. (Elmas, 2011)

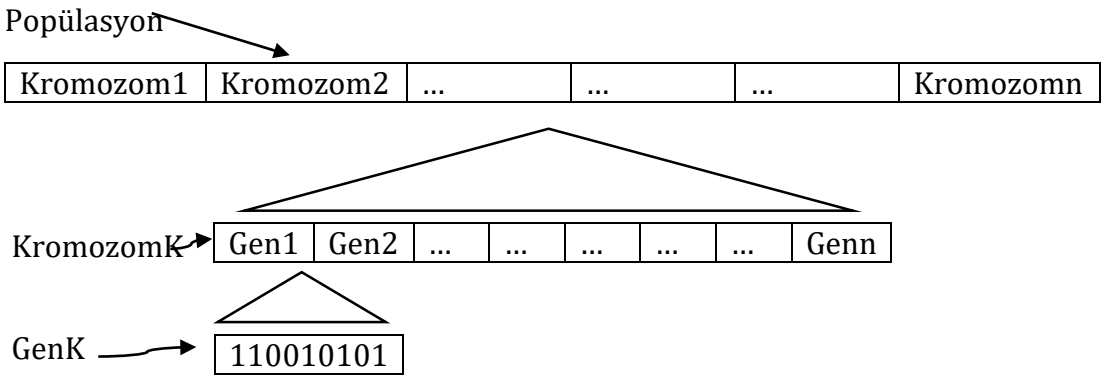
4.3.3. Popülasyon (Yığın)

Popülasyon, çözüm bilgilerini içeren kromozomların bir araya gelmesiyle oluşan olası çözüm yığınına denir. Yığındaki kromozom sayısı sabit olup problemin özelliğine göre programlayıcı tarafından belirlenir. Genetik algoritmanın çalışması esnasında bu yığın kümesinden bir takım kromozomlar yok olmakta ve yerlerine yeni kromozomlar eklenerek popülasyon büyüklüğü sabitlenmektedir.

Yığın büyüklüğü, problemin çözüm süresini etkilemektedir. Fazla sayıdaki kromozom yığını problemin çözüm süresini uzatırken, az sayıdaki yığın çözüm değerlerine ulaşılmamasına neden olabilir. Problemin özelliğine göre seçilecek olan yığın sayısı programcı tarafından iyi belirlenmelidir.

Popülasyonun büyüklüğü yaygın olarak 30 ile 100 adet arası kromozom içerecek şekilde düzenlenmektedir. Popülasyon büyüklüğü problemin tipine göre ve programı yazan kişiye göre daha az ya da daha fazla olabilir.

Şekil 4.1'de gen, kromozom ve popülasyonu gösteren yapı gösterilmiştir.



Şekil 4.1. Popülasyonun yapısı (Michalewicz, 1996)

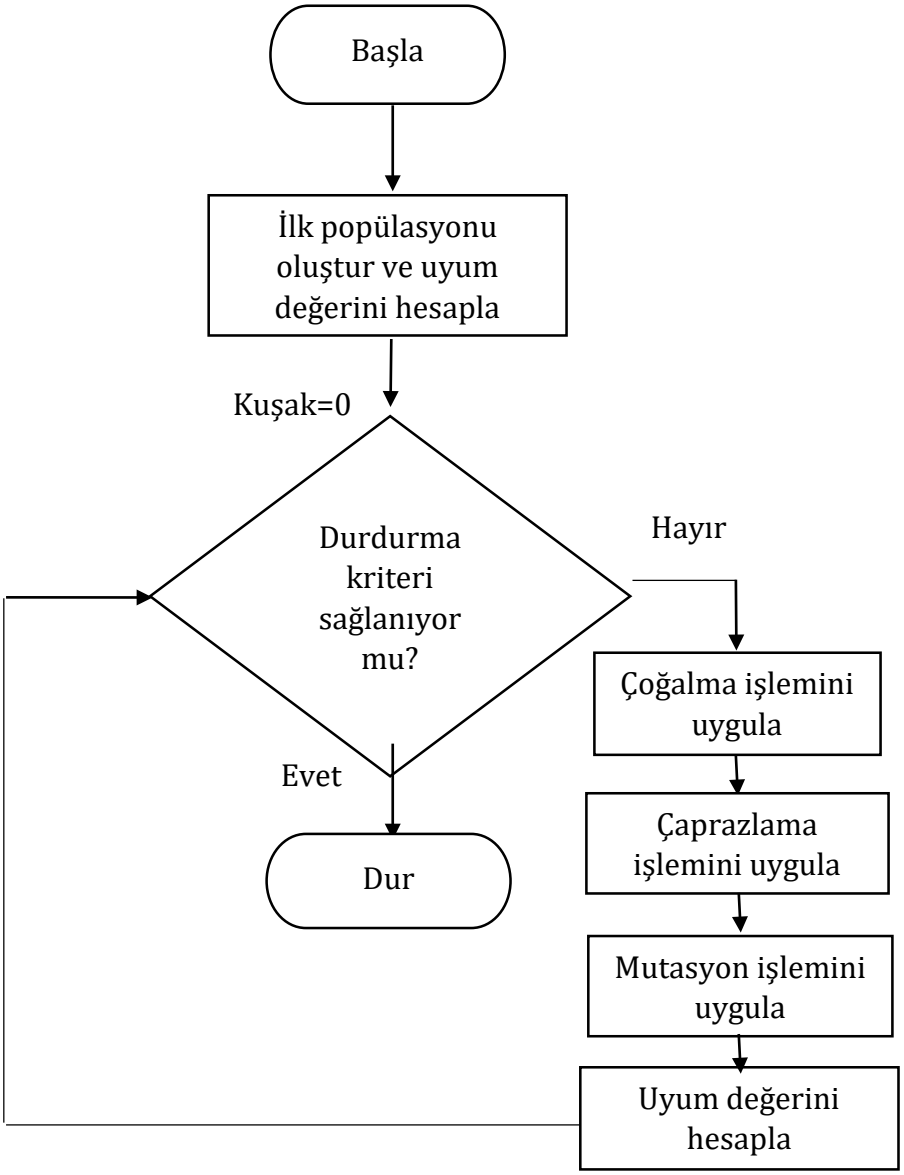
4.4. Çalışma Prensibi

Genetik algoritmalar genel anlamda, dizilerden oluşan bir popülasyona çoğalma, çaprazlama ve mutasyon operatörlerinin uygulanmasını içerir. Bu operatörlerin uygulanmasından sonra yeni bir popülasyon oluşur. Yeni popülasyon eski popülasyon ile yer değiştirir. Her dizinin bir uyum değeri mevcuttur. Diziler uyum değerlerine göre seçilirler. Ortalama uyum değerinin üzerinde uyuma sahip dizilerin gelecek kuşaklarda temsil edilme olasılığı daha yüksektir. Evrim süreci, popülasyonun ortalama uyumunu giderek arttırır ve ilerleyen kuşaklarda daha iyi uyum değerleri edilmesini sağlar.

Genetik algoritmaların ilk adımı, ilk popülasyonun oluşturulup, uyum değerinin hesaplanmasıdır. Daha sonra mevcut popülasyona, temel genetik operatörler

(çoğalma, çaprazlama ve mutasyon) uygulanır. Her kuşak için uyum değeri hesaplanır. Bu durum durdurma kriteri sağlanana kadar devam eder. Şekil 4.2'de genetik algoritmanın akış diyagramı verilmektedir. Genetik algoritmanın adımları şu şekilde açıklanabilir.

- Arama uzayındaki tüm olası çözümlerden bir grup çözüm dizi olarak kodlanır.
- Genellikle rastgele bir çözüm kümesi seçilir ve başlangıç popülasyonu olarak kabul edilir.
- Her bir dizi için bir uyum değeri hesaplanır. Bulunan uyum değerleri dizilerin çözüm kalitesini gösterir.
- Bir grup dizi belirli bir olasılık değerine göre rastlantısal olarak seçilir.
- Yeni dizilerin uyum değerleri hesaplanarak, çaprazlama ve mutasyon işlemlerine tabi tutulur.
- Oluşan yeni popülasyon eski popülasyon ile yer değiştirir.
- Belirlenmiş durdurma kriteri sağlanana dek yukarıdaki işlemler devam ettirilir.
- Döngü, durdurma kriteri sağlanınca sona erdirilir. Amaç fonksiyonuna göre en uygun olan dizi seçilir.



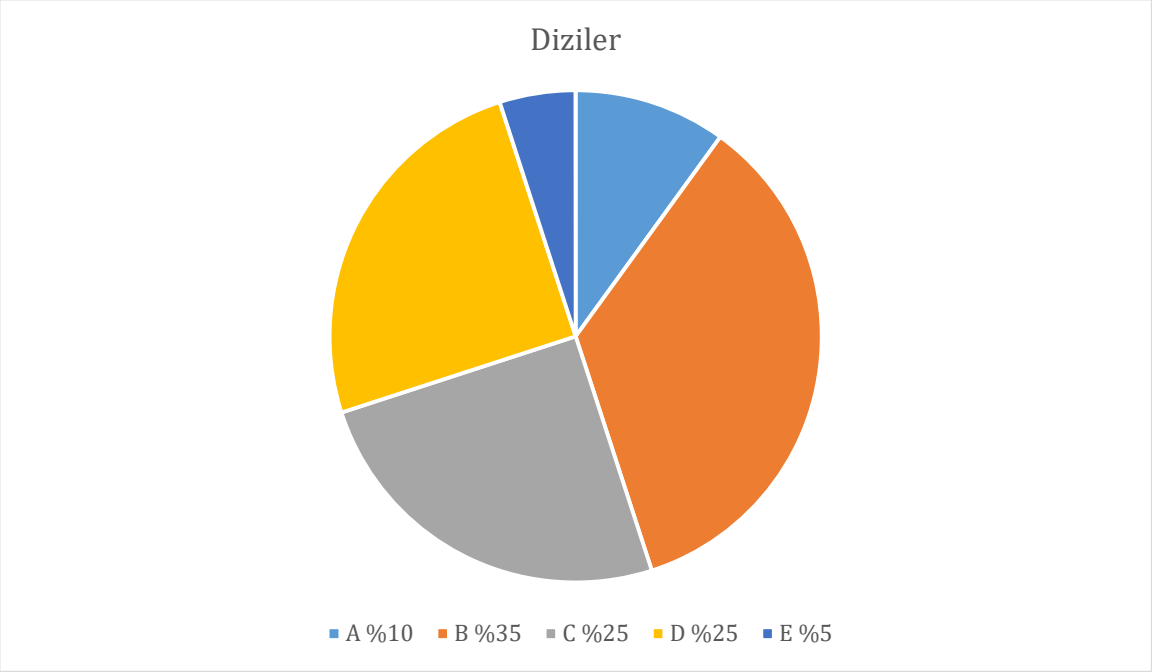
Şekil 4.2. Genetik Algoritma Akış Diyagramı (Taşkın ve Emel, 2009)

4.5. Seçim Mekanizmaları

Bir nesildeki dizilerden bir kısmının bir sonraki nesle aktarılırken bir kısmı da yok olur. İşte bu aşamada hangi dizilerin bir sonraki nesle aktarılacağı kurulan seçim mekanizmaları ile sağlanır. Genetik algoritmalarda kullanılan en basit ve en yaygın olan seçim mekanizması rulet tekerleği (çemberi) seçimidir.

Bu seçimde çember n adet parçacığa bölünmüştür. Her aralık bir diziyi temsil eder. Her dizinin uygunluk değeri toplam uygunluk değerine bölünür. Böylelikle

yığın içindeki her dizinin çözüm kümesi içindeki [0-1] değerleri arasındaki yeri bulunur. Diziler uygunluk değerlerine göre yüzdelik olarak çemberde temsil edilir. Tekrar üreme için rulet tekerleğinin döndürülmesi gerekir. Bunun için sıfırla toplam uygunluk arasında rastgele bir sayı üretilerek bu sayının tekerleğin hangi parçasına karşılık geldiğine bakılarak kromozom seçilir. Böylelikle çemberin bir defa döndürülmesiyle bir sonraki nesle aktarılacak olan dizilerden bir tanesi seçilmiş olur. Benzer şekilde diğer kromozomların da belirlenmesi ile uygunluk değerleri en başarılı olan bireyler eşleştirme havuzuna (mating pool) alınır. Bundan sonra artık diğer nesle ait diziler elde edilir ve genetik operatörlerin uygulanmasıyla yeni nesil elde edilir. Aynı işlem her döngüde devam ederek nesil devamı sağlanır. Şekil 4.3'de örnek bir çizim gösterilmiştir.



Şekil 4.3. Rulet Tekerleği seçme operatörü

Rulet tekerleği yöntemi, basit ve kullanışlı olmasına karşın hataya sahiptir. Bu hata yeni yığında her dizinin beklenen kopya sayısı ile gerçekleşen kopya sayısı arasında büyük farkın olmasıdır. Her bir döngüdeki bu hata programın çözüm değerlerini farklı yönlerde aramasına neden olur. Bu da algoritmanın zamansız yakınsamasına neden olabilmektedir. Bu hatayı azaltmak için bazı araştırmacılar

en iyi bireyi bir sonraki nesle aktarmak için elitizm ve crowding yöntemin gibi çeşitli seçim tekniklerini önermişlerdir. (Altıparmak vd., 1998)

4.5.1. Orantılı Seçim Mekanizmaları

Orantılı seçim mekanizmaları; rulet çemberi mekanizması, rasgele artan seçim mekanizması ve rasgele evrensel seçim mekanizmasıdır. Aşağıda rasgele artan ve rasgele evrensel seçim mekanizmaların kısaca tanımları verilmiştir.

Rasgele artan seçim mekanizması; bu mekanizmada öncelikle yığındaki dizilerin beklenen kopya sayısı hesaplanır. Her dizinin beklenen değerinin tamsayı kısmı kadar kopyası yeni yığına alınır. Yığın genişliğine ulaşılmadıysa yığını doldurmak için beklenen değerlerin kesirli kısımları olasılık olarak yorumlanır. Örnek olarak bir dizinin kopyasının beklenen değeri 1.25 ise bu dizinin bir kopyası alınırken diğer kopyasının alınma olasılığı % 25 olur.

Rasgele evrensel seçim mekanizması; bu mekanizma rulet çemberi mekanizmasına benzemektedir. En önemli farkı çemberin dış kısmının da eşit parçalara bölünmesidir. Bu parçaların sayısı yığının genişliğine eşittir. Seçim aşamasında çember bir kere döndürülür. Bir dizinin kopya sayısı çemberin dış kısmındaki parça sayısı ile belirlenir. Bu durumda bir dizinin çemberdeki ağırlık değerleri verilmiş olan aralığına düşen parça sayısı o dizinin kopya sayısını verir. (Michalewicz, 1996)

4.5.2. Sıralı Seçim Mekanizmaları

Yığındaki diziler uygunluk değerlerine göre iyiden kötüye doğru sıralanırlar. En iyi diziden başlanarak azalan bir işlev yardımıyla dizilere kopya sayısı belirlenir. Kullanılan en genel atama işlevi doğrusaldır. Bir fonksiyon yardımıyla atanan kopya sayıları yeni yığının oluşturulmasında kullanılır. Bu aşamada orantılı seçim mekanizmalarından birisi kullanılarak yeni yığın elde edilir. (Elmas, 2011)

4.5.3. Turnuva Seçim Mekanizması

Yığından rasgele bir grup dizi seçilir. Grup içindeki en iyi uygunluk değerine sahip dizi yeni yığına kopyalanır. Yığın genişliğine ulaşıncaya kadar bu işlem devam eder. (Altıparmak vd., 1998)

4.5.4. Denge Durumu Seçim Mekanizması

Anlatılan diğer seçim mekanizmalarında mevcut yığından yeni diziler seçilerek yeni yığın oluşturulur. Oluşturulan bu yığındaki bireylere genetik operatörler uygulanarak yeni diziler elde edilir. Elde edilen bu dizilerden seçim yapılır. Denge durumu seçim mekanizmasında ise doğrusal seçim mekanizması kullanılarak seçilen birkaç adet bireye genetik operatörler uygulanır. Elde edilen diziler mevcut yığındaki uygunluk değeri düşük olan bireylerle yer değiştirilir. (Elmas, 2011)

4.6. Genetik Operatörler

Genetik algoritmada çözüm yığını incelenirken belirli noktalardan sonra nesil çeşitliliği olmadığı için çözüme gidilememektedir. Nesil çeşitliliğini sağlayarak çözüm uzayında algoritma istenen kısıtları sağlayacak olan çözüm yığınına ulaşabilir. Bunun için dizileri çaprazlama (crossing over) ve değişim (mutation) operatörleri belirli yüzdelik oranlarıyla uygulanarak nesil çeşitliliği sağlanır. Böylelikle sistemin belirli noktalara gelip takılması önlenmiş olur. Aşağıda bu genetik operatörlerin çeşitleri ve uygulanışları anlatılmıştır.

4.6.1. Çaprazlama Operatörü

İki dizinin bir araya gelerek karşılıklı gen yapılarının değişimi ile yeni dizilerin oluşumunu sağlayan operatördür. Çaprazlanarak gen (bilgi) değişiminin yapılmasından önce dizilerin çaprazlamaya tutulma olasılığı belirlenmelidir. Bu oran %50-%95 oranında uygulanmaktadır. Çaprazlamada bir diğer önemli unsur ise ne tür bir çaprazlamanın yapılacağıdır. Mesela eş kromozom seçiminde ilk

kromozom en yüksek uygunluk değerine sahip kromozom seçilirken ikinci kromozom rasgele olarak seçilebilir.

Bir yığına çaprazlama operatörü p_c olasılığı ile uygulanır. Çaprazlama oranı, çaprazlama operatörünün kullanım sıklığını kontrol eder. p_c .l.N adet kromozoma çaprazlama uygulanır. Yüksek çaprazlama oranı, popülasyon değişkenliğini hızlı bir şekilde gerçekleştirir. Düşük çaprazlama oranı, aramanın çok yavaş gerçekleşmesine sebep olur.

Tek noktalı çaprazlama operatörü; bu operatörde çaprazlama noktası 1 ile L-1 arasında rasgele seçilir. Eşlenen iki dizide bu çaprazlama noktasından sonraki bölümler yer değiştirerek iki tane yeni birey elde edilir. Şekil 4.4'de tek noktalı çaprazlama için örnek bir uygulama yapılmıştır.

1.Ebeveyn	10110	01001	➡	1.Çocuk	10110	**11010**
2.Ebeveyn	**11000**	**11010**	➡	2. Çocuk	**11000**	01001

Şekil 4.4. Tek Noktalı Çaprazlama

Çok noktalı çaprazlama operatörü; bu operatörde çaprazlama noktası 1 ile L-1 arasında rasgele çoklu bölge seçilir. Eşlenen iki dizide bu çaprazlama noktaları arasında kalan bölümler yer değiştirerek iki tane yeni birey elde edilir. Şekil 4.5'de örnek üzerinde incelenmiştir.

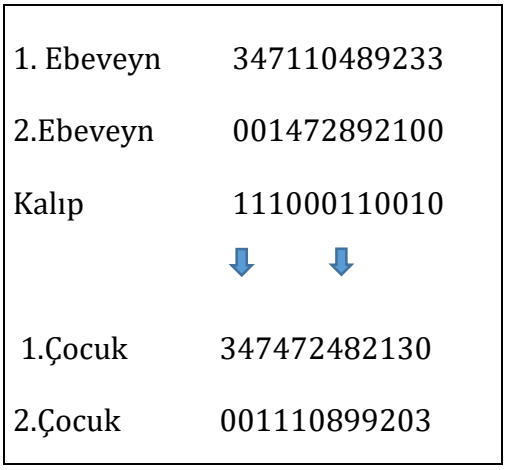
1.Ebeveyn	10 110	01 001	➡	1. Çocuk	10 **000**	01 **010**
2. Ebeveyn	**11 000**	**11 010**	➡	2.Çocuk	**11** 110	**11** 001

Şekil 4.5. Çok noktalı Çaprazlama

Tek noktalı ve çok noktalı çaprazlama işlemi Genetik algoritmada ilk akla gelen çaprazlama yöntemleridir. Ancak problemin özelliğine göre farklı tiplerde çaprazlama yapmak da mümkündür. Bu çaprazlama yöntemlerinden birkaç tanesini kısıtlı eniyileme problemleri için incelenerek olunursa aşağıda belirtilen tiplerde çaprazlama yapmak mümkündür. Atölye çizelgeleme gibi kısıtlı eniyileme problemlerinde, gen kodlamanın farklı olmasından ve probleme uygun çeşitli çaprazlama türleri üzerinde çalışmalar yapılmıştır. Bunlardan bazıları;

- Pozisyona dayalı çaprazlama
- Sıraya dayalı çaprazlama
- Kısmi planlı çaprazlama

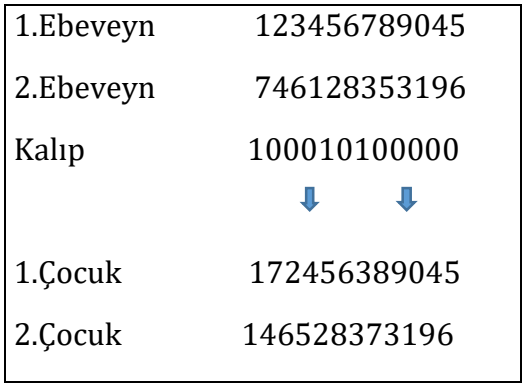
Pozisyona dolayı çaprazlama; bu çaprazlamada kalıp olarak, sabit kalacak olan gen yapılarını belirlemede kullanılan yapı bulunur. Kalıbın gösterdiği noktalar dizide sabit kalırken diğer noktalar iki birey arasında yer değiştirilerek yeni bireylerin oluşumu sağlanır. Şekil 4.6'da gösterildiği gibi kalıp dizisinde 1'lerin gösterdiği değerler sabit kalacak değerleri göstermektedir.



Şekil 4.6. Pozisyona dayalı çaprazlama

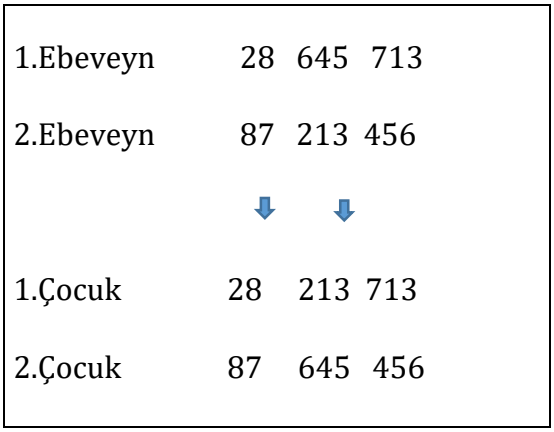
Sıraya dayalı çaprazlama*;* Şekil 4.7 'deki örnekten de görüldüğü gibi; kalıp üzerindeki 1'lerin gösterdiği değerler çaprazlamada kullanılacak olan değerleri belirtir. A2'den sırasıyla 7, 2, 3 değerleri çaprazlanacak olan genlerdir. A1'de

bulunan 2, 3, 7 değerleriyle aynı sıralı olacak şekilde yer değiştirilir. Aynı işlem 1'lerin A1'de gösterdiği değerlerin A2'ye aktarılmasıyla tamamlanır.



Şekil 4.7 Sıraya dayalı çaprazlama

Kısmi planlı çaprazlama; iki bireyden rasgele bir aralık belirlenir. Bu aralıktaki değerler yer değiştirilir. Şekil 4.8.'de bu çaprazlama gösterilmiştir.



Şekil 4.8. Kısmi planlı çaprazlamada 1.adım

Yer değiştirme sonunda dizide aynı olan değerler değiştirilen değerlerle tamamlanır. Şekil 4.9'da gösterilmiştir.

1.Çocuk	68	213	745
2.Çocuk	87	645	123

Şekil 4.9. Kısmi planlı çaprazlamada 2. Adım

Önceden de bahsedildiği gibi problemin özelliğine göre farklı yapılardaki çaprazlama yöntemleri kullanılabilir. Ancak temel olarak tek ve çok noktalı çaprazlama yöntemleri kullanılmaktadır. (Michalewicz, 1996)

4.6.2. Değişim (Mutasyon) Operatörü

Genetik algoritmada sistem belli döngü değerine geldikten sonra diziler birbirine giderek benzemektedir. Bu da çözüm uzayının daralmasına neden olmaktadır. Dizilere ne kadar çaprazlama yöntemi uygulansa da ilerleyen nesillerde dizi çeşitliliği sağlanamamaktadır. Bu durumda dizinin kendi içindeki genler rasgele yer değiştirilir. Böylelikle dizi çeşitliliğinin devamı sağlanmış olur. Ancak değişim operatörünün uygulanma oranı doğru belirlenmelidir. Değişim oranının yüksek olması çözüm uzayını çok genişleterek sistem çözümünün yanlış yerlerde oranlanmasına neden olur. Bu nedenler değişim operatörünün uygulanma olasılığı %0.5-%15 arasında değiştirmektedir.

Özellikle Genetik algoritmanın ilerleyen nesillerinde değişim etkinliği artmaktadır. Çünkü ilerleyen nesillerde popülasyon iyi çözümlere yakınsandığından, kromozomlar birbirine çok benzemektedir. Bu durum çaprazlama operatörünün aramasını kısıtlar. Nitekim çaprazlama sonucu elde edilen kromozomlar birbirine benzer olacaktır. Bu aşamada değişim operatörü, popülasyondaki değişkenliği gerçekleştirerek arama uzayında yeni çözüm noktalarının elde edilmesini sağlanmaktadır.

Değişim işlemi, p_m olasılığı ile tek bir pozisyonun rasgele değişimi olup bu işlem oluşturulmuş neslin elverişli durumunu birden bozabileceği için önemlidir.

Sonuçta p_m .N adet değişim gerçekleşir. Böylece p_m olasılığı küçük tamamlanır. Şekil 4.10'da örnek üzerinde gösterilmiştir.

	Değişim Öncesi		Değişim Sonrası
1.Çocuk	10110 **1** 1010	➡	1.Çocuk 10110 **0** 1010

Şekil 4.10. Değişim operatörünün uygulanması

Şekil 4.10.'da belirtilen değişim operatörünün basitçe gösterimidir. Değişim operatörünün uygulanma biçimi genetik algoritmanın kullanıldığı probleme göre değişebilir. Aşağıda kullanılması muhtemel değişim operatörlerinden bir kaçı gösterilmiştir.

Komşu iki işi değiştirme; Şekil 4.11.'de görüldüğü gibi rasgele seçilen iki komşu gen yer değiştirilir.

Değişim Öncesi		Değişim Sonrası
1.Çocuk	257***19***432	➡ 1.Çocuk 257***91***432

Şekil 4.11. Komşu iki genin değişimi

Keyfi iki işi değiştirme; Şekil 4.12.'de görüldüğü gibi rasgele seçilen iki gen değiştirilir.

Değişim Öncesi		Değişim Sonrası
1.Çocuk	2***5***719***4***32	➡ 1.Çocuk 2***4***719***5***32

Şekil 4.12. Keyfi iki genin değişimi

Keyfi üç iş değiştirme; Şekil 4.13.'de görüldüğü gibi rastgele seçilen üç gen rastgele yer değiştirilir.

Değişim Öncesi		Değişim Sonrası	
1.Çocuk	2***5***7***1***9***4***32 ⟹	1.Çocuk	2***4***7***5***9***1***32

Şekil 4.13. Keyfi üç genin değişimi

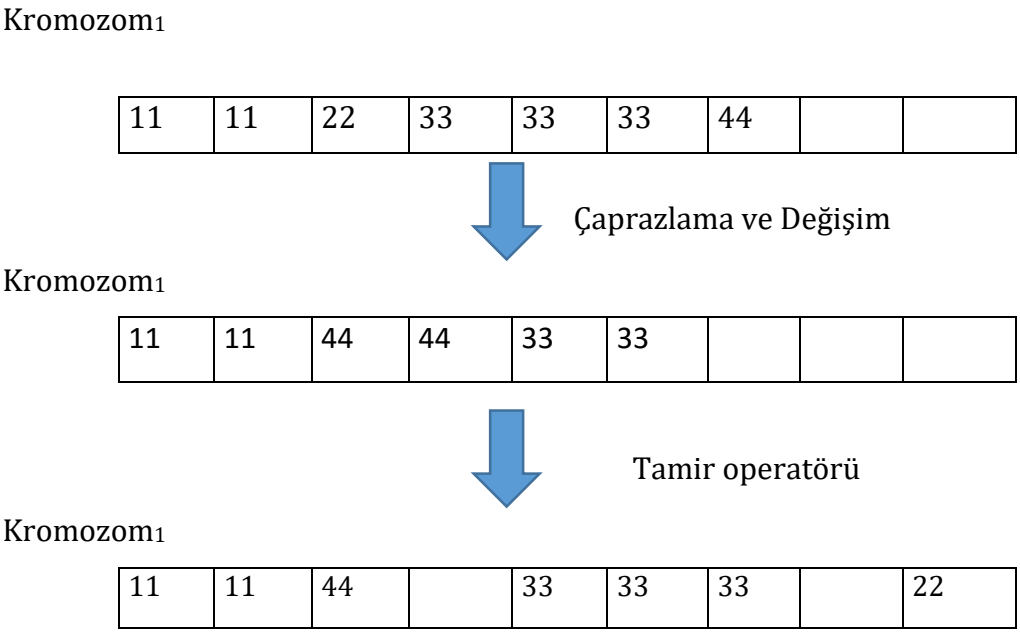
Araya gen ekleyerek değiştirme; Şekil 4.14 görüldüğü gibi keyfi olarak seçilen genin rasgele sayıda sağa veya sola kaydırılmasıyla gerçekleştirilir.

Değişim Öncesi		Değişim Sonrası	
1.Çocuk	257194***3***2 ⟹	1.Çocuk	2***3***5719432

Şekil 4.14. Kaydırmalı gen değişimi (Tiwari ve Vidyarthi, 2000)

4.6.3. Tamir Operatörü

Tamir operatörü (düzenleyici algoritma); uygun olmayan dizileri uygun duruma getirmek için özel olarak tasarlanan algoritmadır. Problemin özelliğine göre geliştirilen bu algoritmada genetik operatörlerin uygulanmasından sonra diziden mevcut bilgilerin yok olması veya fazladan istenmeyen bilgilerin gelmesi çizelgeleme problemlerinde istenmeyen bir durumdur. Bu sorunu ortadan kaldırmak için başlangıç dizisinin bilgilerine sağdık kalarak özel bir algoritma geliştirilir.



Şekil 4.15. Genetik işlem sonrası dizi durumu

Şekil 4.15'te görüldüğü gibi genetik işlem sonrası oluşan yeni dizide fazla ve(veya) kaybolan genler bulunmaktadır. Bu dizinin tamir edilip bir sonraki nesle aktarılarak genetik algoritmanın çalışmasına devam edilir. Bu yaklaşımda probleme özgü genetik operatörler kullanılır. Amaç genetik operatörler sonucunda elde edilen yeni dizilerin uygun birer çözüm kümesini içermesidir. (Davis, 1991)

4.6.3. Elitizm (En İyinin Saklanması) Yöntemi

Elitizm ya da en iyinin saklanması olayında yığın içindeki en iyi bireylerin ya da belli genişlikteki yüzdeliğe sahip bireylerin o yığından alınarak hiçbir değişikliğe uğratılmadan bir sonraki nesil yığınına aktarılır. Genetik operatörlerin kullanımı sonrası en iyi bireyin yok olması söz konusu olduğu için yığın içindeki çözümü en iyi temsil eden dizi bir sonraki nesle kopyalanır.

Genetik algoritmanın temel kavramlarının açıklamasından sonra, genetik algoritmanın problemi çözme aşamasında yapısının nasıl oluşturulduğunu basamaklar şeklinde örnek bir uygulamada anlatmak, konunun daha iyi anlaşılmasını sağlayacaktır. (Elmas, 2011)

4.7. Uygulama Alanları

Genetik algoritmalar, özellikle son yıllarda, karmaşık problemlerin hızlı ve kabul edilebilir sonuçlar elde edilerek çözümüne imkan sağlamıştır. Karmaşık problemlerde geleneksel yöntemlerle çözüm, genetik algoritmalara göre daha uzun sürebilmektedir. Genetik algoritmalar özellikle bu tip problemlerde iyi bir alternatif olmaktadır. (Gonzales ve Fernandez, 2000)

Genetik algoritmalar geniş bir kullanım alanına sahiptir. Makina öğrenmesi, veri madenciliği uygulamaları ve bilgisayar ağ tasarımı bilgi sistemlerindeki en yaygın kullanımıdır. Bunun dışında optimizasyon problemlerinin çözümünde, özellikle montaj hattı dengeleme, çizelgeleme, tesis yerleşim, hücresel üretim, araç rotalama ve atama problemlerinde kullanılmıştır.

4.7.1. Genetik Algoritma ile Optimizasyon

Genetik algoritmaların kullanım alanlarından biri de optimizasyon problemleridir. Maksimize ve minimize problemlerinde kullanımı mümkündür. Genetik algoritma ile bir optimizasyon probleminin çözülmesinde en büyük zorluk problemi genetik algoritma ile çözmeye uygun hale getirmektir. Fonksiyon bu hale getirildikten sonra çözüm uygunluk fonksiyonunun belirlenmesi, mutasyon ve çaprazlama teknikleri uygulanarak çözülebilmektedir.

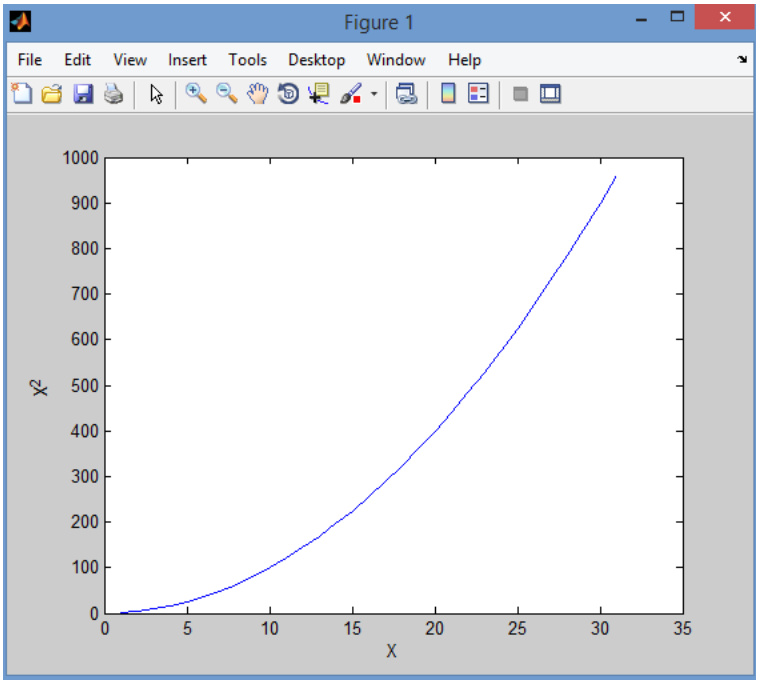
Basit bir tek değişkenli fonksiyonun optimizasyonu ile genetik algoritmayı ele alalım. Genetik algoritma çalışmalarında sıklıkla kullanılan $F(x) = x^2$ fonksiyonunun maksimize edilmesini inceleyelim. Bu örnek ile genetik algoritmanın nasıl çalıştığı detayları ile anlatılmıştır.

Fonksiyon $F(x) = x^2$ olup, $0 \leq x \leq 31$ tam sayı aralığında fonksiyonu maksimum eden değerin bulunması istenmektedir. Şekil 4.16 'da fonksiyonun grafiği verilmiştir. Açıkça görüldüğü gibi $x = 31$ değeri için fonksiyon maksimum değerini verecektir.

Grafiği çizmek için aşağıdaki kod satırı matlab programında yazılır.

```
>> x = [0:31];
>> y = x.^2;
>> plot(x,y);
>> xlabel('X');
>> ylabel('X^2');
```

Yazılan kodlar çalıştırıldığında şekil 4.16'da ki gibi bir grafik görüntülenir.



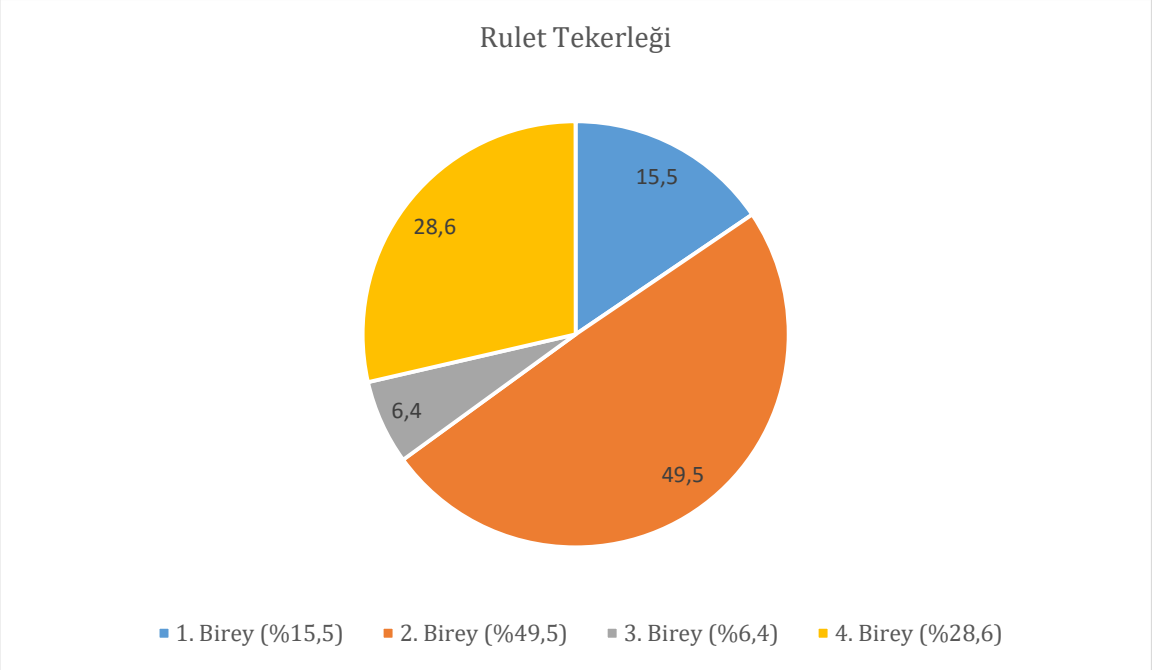
Şekil 4.16. F(x) = x^2 fonksiyonu

Problemin çözümü için ilk önce x değişkenini, uzunluğu 5 bit olan işaretsiz ikili tam sayı olarak kodlayalım. Bu sayıları rastgele seçiyoruz. Seçim için bir para atılıp yazı gelen tarafı 1, tura gelen tarafı 0 olarak alınabilir. Böylelikle 4 boyutlu popülasyon olarak tasarlanan örneğimizin ilk popülasyonunun kromozomları rastgele 5 bitlik dizilerden oluşturulmuş ve çizelge 4.2. 'de gösterilmiştir.

Çizelge 4.2. Rastgele oluşturulmuş başlangıç popülasyonu

Bireyler (i)	**Kromozomlar**	**Değer** (x)	**Uygunluk Değeri** $f_i(x) = x^2$	**Toplam (%)**
1	01110	14	196	15,5
2	11001	25	625	49,5
3	01001	9	81	6,4
4	10011	19	361	28,6

Başlangıç popülasyonundaki bireylerin kromozomları ile fonksiyonun ürettiği değerler çizelge 4.2. 'de verilmiştir. Her bir kromozom fonksiyona girerek bireylerin uygunluk değeri hesaplanmış, uygunluk değerlerinin toplamı her bireyin uygunluk değerine oranlanarak yüzde değerleri hesaplanmıştır. Bu değerler kullanılarak rulet tekerleği yöntemi ile üreme işlemi yapılacaktır. Bireylerin rulet tekerleğinde gösterimi şekil 4.17'de gösterilmiştir.



Şekil 4.17. Rulet tekerleği gösterimi

Rulet tekerleğindeki paylara göre bireylerin bir sonraki nesle aktarımı gerçekleşecektir. Çizelge 4.3. 'de görüleceği gibi 1 ve 4. bireylerin birer, 2. bireyin ise iki kopyası bir sonraki nesle aktarılmıştır. 3. birey bir sonraki nesle aktarılmayacaktır.

$\sum f_i = 1263$, $f_{ort} = 1263 / 4 = 315{,}75$

Çizelge 4.3. Üreme işlemi

Birey (i)	**Kromozom**	**Uygunluk değeri** $f_i(x) = x^2$	**Seçim Değeri** $P_i = f_i / \sum f_i$	**Beklenen** f_i / f_{ort}	**Kopya Sayısı**
1	01110	196	0,155	0,620	1
2	11001	625	0,494	1,979	2
3	01001	81	0,064	0,257	0
4	10011	361	0,285	1,143	1

Çizelge 4.3. ‘de üreme işleminin detayları verilmiştir. Bu değerler ile çaprazlama işlemine geçilir. İlk önce popülasyondaki yeni bireyler rastgele eşleşir. Daha sonra eşleşen bireylerin bit değişimi için yine rastgele çaprazlama pozisyonu belirlenir. Belirlenen kıstaslarda çaprazlama işlemi gerçekleştirilir. Çizelge 4.4. ‘de bu işlemin detayları verilmiştir. Buna göre belirlenen çaprazlama pozisyonları ile 1 ve 2. bireyler ile 3 ve 4. bireyler eşleştirilmiştir.

Çizelge 4.4. Çaprazlama işlemi

Çaprazlama Öncesi	**Çaprazlama Sonrası**	**Değer** (x)	**Uygunluk Değeri** $f_i(x) = x^2$	**Seçim Değeri** $P_i = f_i / \sum f_i$	**Beklenen** f_i / f_{ort}
0111 | 0	01111	15	225	0,124	0,495
1100 | 1	11000	24	576	0,317	1,267
110 | 01	11011	27	729	0,401	1,603
100 | 11	10001	17	289	0,159	0,636

Çaprazlama sonrası oluşan değerlerin toplam ve ortalaması şu şekildedir:

$\sum f_i = 1819$, $f_{ort} = 1819 / 4 = 454{,}75$

Çizelge 4.4. ’de görülen yeni popülasyonun 3. bireyinin 11**0**11 kromozomuna mutasyon uygulayalım. Mutasyon işlemi sonucunda yeni değer 11**1**11 olacaktır. İkili kodda 11111 değerinin ondalık karşılığı 31 ‘dir.($(11111)_2 = (31)_{10}$) Bu da

bize $0 \leq x \leq 31$ tam sayı aralığında fonksiyonu maksimum eden $x = 31$ değerini verecektir.

5. PROJE YÖNETİMİ

5.1. Proje ve Proje Yönetimi

Proje, belirli bir amaca yönelik önceden belirlenmiş sonuçlara ulaşabilmek için yapılan çalışmalar olarak ifade edilebilir. Proje yönetimi ise belirlenen bütçe, kalite ve zaman kısıtlarına uygun olarak tüm kaynakların (insan, makine, materyal vb.) planlanması, yönetilmesi ve kontrol edilmesi işlevidir.

5.2. Proje Yönetimi Yöntemleri

Projeler hangi konuda olursa olsun bir ekip işidir. Projelerde ekip üyeleri için özelleşmiş rol ve görevler bulunur. Tek bir kişinin projede bulunan tüm bu farklı görevleri yapması mümkün değildir. Bu durum kişilerin birlikte çalışması ihtiyacını doğurur. Her rol ve görev için konusunda uzman kişilerin çalışması önemlidir. Yönetim bu durumda ekibin organizasyon ve koordinasyonunu sağlamak olarak düşünülebilir. Projelerin zamanında, bütçesinde ve kalitesinde bitirilebilmesi ancak sağlıklı bir yönetimle mümkün olmaktadır.

Yönetim bir amaç etrafında ekibin faaliyetlerinin bütünlüğü ve ekip içi iletişimi sağlama, görevlendirme yapma ve yapılan çalışmaların amaca uygunluğunu değerlendirme faaliyetleri olarak tanımlanabilir. Yönetmek analiz, tahmin ve plan gibi analitik faaliyetlere dayanır. (Güney, 2007)

Proje yöneticisi projenin başarı ve performansından sorumludur. Bu kapsamda karar verme ve görevlendirme yetkilerine sahiptir. Yönetim zor ve çok yönlü bir alandır. Birçok faklı beklenti ve teknik/idari/mali kısıt arasında denge kurma gibi oldukça zor bir görev proje yöneticisinin omuzlarındadır. Tüm bunların üstesinden gelebilmek sadece teknik bilgi ile mümkün değildir. Yönetimin temel ilkelerini bilerek hareket etmesi gerekir. Proje yöneticisi, yönetim bilgisi ile üst yönetimin herhangi bir konuda nasıl hareket edeceğini ve projenin kurumun hedeflerini başarmasına nasıl katkı sağlayacağını anlayabilir. Bu anlayış, üst

yönetici, ekip üyeleri ve kullanıcılarla doğru şekilde iletişim kurmak için hayati öneme sahiptir.

5.2.1. Genel Yönetim Akımı

Henry Fayol, birçok kaynakta yönetim ilminin kurucusu kabul edilir. Yönetim fonksiyonlarını öngörmek, organizasyon, liderlik, koordinasyon ve kontrol olarak açıklamıştır. Yönetim için 14 temel ilke önermiştir. (Vishnu, 2007)

- işbölümü: İş ve yöntemlerin özelleşmesiyle sürekli gelişme ve üretkenlik artışı sağlanacağı ileri sürülür.
- Otorite: Yöneticinin otoriteden gelen emir verme hakkı ve itaat sağlama gücü olmalıdır. Otorite sorumluluk da gerektirir.
- Disiplin: Bir çalışan kurallara uyma, boyun eğme ve kuruma saygı çerçevesinde disiplinli hareket etmelidir. Disiplin; liderlik, kuralların anlaşılması ve cezalandırma ile sağlanır.
- Kumanda birliği: Bir çalışanın sadece ve sadece bir tek yöneticisi olmalıdır.
- Yürütme birliği: Planı bir kişi yapmalı diğerleri uygulamalıdır.
- Özel çıkarların geri plana itilmesi: İşte sadece iş düşünülmeli ve özel ilgiler kurumsal görevlerin önüne geçmemelidir.
- Ücretlendirme: Kişi izinsiz yaptığı işler için değil, kendisinden istenilenleri yapması karşılığı ücret almalıdır.
- Merkeziyet: Yönetim bütünleşmeli ve karar yukarıda alınmalıdır.
- Hiyerarşi: Komutların yukarıdan aşağı aktığı yönetim örgütlenmesi önerilir.
- Düzen: Her malzeme ve kişi için bir seviye (yer ve zaman) tanımlanmalı ve o noktada kalınmalıdır.
- Eşitlik: Çalışanlara eşit muamele edilmelidir.

- Süreklilik: İyi çalışanlar için yaşam boyu iş güvencesi sağlanmalıdır.

- İnsiyatif: Çalışanların planda kastedilenleri anlama ve yapma insiyatifi olmalıdır.

- Çalışanlar arasında birlik: Ekip ruhu ve birlik için teşvik, buna uymayan davranışların engellenmesi ve komutların ekip olarak uygulamasının sağlanmasıdır. (Fayol, 2012)

Bu ilkelerin bir kısmı günümüzde bir askeri birliğin çalışma talimatı gibi düşünülebilir. Ancak bugün dahi birçok üst yöneticinin uygulamaya çalıştığı gözlemlenmektedir.

5.3.2. Bilimsel Yönetim

Modern yönetim biliminin kurucusu olarak kabul edilen Amerikalı mühendis Frederick W. Taylor 1911 yılında bilimsel yönetimin ilkelerini ortaya koymuştur. Taylor daha çok üreten kişilerin az üretenlerle bir tutulamayacağından hareketle bir iş yapma ve kontrol sistemi geliştirmiştir. Tek bir işçinin üretkenliğini artırmak için bu işçinin yaptığı her işin parça parça incelenmesini önermiştir. Uzun ve daha sıkı çalışma yerine, yapılan işi daha etkin yapmanın önemini vurgulamıştır. Bunun dışında yönetim fonksiyonlarının diğer işlerden ayrılması gerektiğini belirtmiştir. Önerdiği temel ilkeler:

- İşlerin geliştirilmesi için bilimsel yöntemlerden faydalanmak
- İşe uygun çalışanın seçiminde bilimsel yöntemlerden faydalanmak
- Çalışanın bilimsel yöntemler doğrultusunda eğitim almasının sağlanması
- Organizasyonda hiyerarşik bir yapının kurulması

Taylor'un çağdaşı Henry Gantt işleri bölme ve sıralamaya dönük çalışarak kendi ismiyle anılan, günümüzde yaygın kullanılan Gantt Şemasını önermiştir. Bu şema ile işler, işler arası bağlantı ve kilometre taşları standart bir gösterime kavuşmuştur. (Nizam, 2014)

5.3.3. Süreç Tabanlı Yönetim

1930'lu yıllarda Shewhart W. üretim sürecinde değişkenlik ve kalite arasındaki bağlantıları istatistiksel olarak inceleyen çalışmalar yapmış ve bu amaçla kontrol grafiğini geliştirmiştir. (Shewhart, 1931) Geliştirme yapılırken süreçte meydana gelebilecek değişkenliği azaltmanın faydasını ifade etmiştir. Elbette her süreçte değişkenlik olacaktır. Bunlar sürecin doğasından kaynaklanan kontrol edilebilir ve süreç dışından kaynaklanan kontrol edilemez değişkenlik olarak sınıflandırılır.

Yine 1930'lu yıllarda Henry Ford yüksek miktarlarda otomobili düşük maliyetle üretebilmek için bir üretim hattı modellemiştir. Bu modelde kalite kontrol çalışmalarına önem verdiği gibi aynı zamanda sadece ürüne odaklanmamış ve üretim sürecine bilimsel olarak yaklaşmıştır. Böylelikle süreç, karmaşık işlerin parçalara bölünerek gerçekleştirilmesine olanak sağlamıştır. Önerilen temel ilkeler:

- Ürünlerin standartlaştırılması
- İşgücünü en aza indiren üretim hattı
- İşçi için iyi ücret

Deming, iş süreçlerinin sürekli geri besleme alınan çevrimlerle yönetilmesini ve iyileştirilecek kısımların bu yöntemle tespitini önerir. Geliştirilen bu teori 1950'lerde Japon sanayinde uygulanmıştır. (Deming, 1986) Deming'e göre kalite, kontrollerle değil kaliteyi hedefleyen iyileştirmelerle sağlanır.

5.3.4. İnsan İlişkileri

Bir dizi deney için 1924-1932 yılları arasında Elton Mayo liderliğinde bir ekip oluşturulmuş ve bu ekip üzerinden farklı yönetim şekilleri ve insan ilişkilerinin, çalışma performansı ve üretkenliğe etkisi gözlemlenmiştir. Bu deneyler ile kişiler arası sosyal iletişimin, çalışma performansını çevre koşullarından daha fazla etkileyebileceğini göstermiştir.

Deneyler için seçilen kişilerde gruplaşmalar olmuştur. Kişiler arasındaki gruplaşmalar gruba uymayanların yalnızlaşmasına neden olmuştur. Deneyler sonucunda ortaya çıkan temel ilkeler (Bagad, 2008):

- Gayri resmi kendi kendine oluşan gruplar güçlü etkilere sahiptir.
- Kişiler tecrit edilmemeli grup olarak görülmelidir.
- Parasal kazanç ve çevre koşulları, gruba aidiyet kadar önemli değildir.
- Yöneticinin kendisiyle ilgilendiğini hissetmek, kişinin çalışma verimini artırır.
- Yönetici, kişinin sosyal ihtiyaçlarının farkında olmalı ve kişinin kurumsal organizasyona karşı değil, kurumla birlikte hareket etmesini sağlamalıdır.

5.3.5. Bilgi Çağında Yönetim

Yönetim 1950'lerden sonra bilim olarak düşünülmeye başlanmıştır. Bundan önceki çalışmalarda yönetim sektörel alanlarla kısıtlı olarak inceleniyordu. Ekonomi, hastane, sosyal kurumlar gibi farklı yapıların yönetimi ayrı ayrı incelenmekteydi. Ancak farklı alanlarda da yapılsa yönetim faaliyetlerinin ortak noktaları olduğu tespit edilmiş ve bu da yönetimi bir bilim dalı haline getirmiştir

Günümüzde bilgi ve teknolojiye dayalı yeni modeller geliştirilmiş ve artık insan gücüne dayalı geleneksel üretim modellerinin yerini almaya başlamıştır. Eskiden yapılan çok fazla üretmeye dayalı ürün modelleri tercih edilmemekte bunun yerine esnek ve hızla değişen ürünlere imkan sağlayacak modeller geliştirilmektedir. Çağımızın yenilikçi ve itici gücü yazılım ve bilişim sistemleridir. Bu sistemlerde çalışacak kaynaklar da iyi eğitim almış yazılım ve bilişim uzmanlarıdır. Yazılım ve bilişim uzmanları, geleneksel model çalışanlarına göre çok daha fazla araştırmak, bilmek ve elbette yeri geldiğinde karar almak durumundadır.

Geleneksel işgücünü verimli kılan ve bilgiyi içinde barındıran sistemlerde sistem üretkendir. Tek tek çalışanların, fazla bilgi ve hüner sahibi olmadan çalışmalarını mümkün kılar. Buna karşılık bilgi tabanlı bir modelde, tüm sistemi başarılı kılan

çalışanların üretkenliğidir. Geleneksel işgücünün bir parçası olan çalışan, sisteme hizmet eder. Buna karşılık bilgi çağında çalışan daha farklı konumlandırılır. Burada sistemin çalışana hizmet etmesi söz konusudur (Drucker, 2006).

5.4. Proje Yöneticisi Görev ve Sorumlulukları

Proje yöneticisi, projenin başarısı ve başarısızlığı konusunda hesap verici konumunda bulunan kişidir. Projenin başarısı için, kişisel özellik, bilgi ve tecrübesini kullanır. Proje yöneticisinin görevleri arasında işleri planlamak, proje ekibini planın doğruluğuna ikna etmek, plana uygun hareket edilmesini sağlamak, oluşabilecek sorunları gidermek, ekibe moral vermek, ekip içi ve dışındaki kişilerle iletişim kurmak bulunmaktadır. Proje yöneticisi anlaşmazlık durumunda nihai kararı alır. Kendisini çözemediği durumlarda varsa yönlendirme komitesini yoksa üst yönetimi bilgilendirir.

5.4.1. Liderlik

Bu çalışma ve daha birçok kaynakta proje yöneticisinin bir yönetici olarak nasıl davranması gerektiği hususu anlatılmaktadır. Ancak tüm proje ve proje ekipleri birbirinden farklıdır. Bu nedenle hiçbir yöntem proje yöneticisinin yeni bir projede karşılaşacağı durumları tam olarak bilemez. Proje yöneticisi karşı karşıya kaldığı bu durumlarda karar verirken yalnızdır. Liderlik işte burada önem kazanmaktadır. Lider gibi düşünmeli ve hareket etmelidir.

Proje yöneticisi insanların saygınlığını ve güvenini kazanabilme, azim, kararlılık, sorumluluk alma, zor zamanlarda soğukkanlılığı korumak, ekibi üzerinde etkili olma, hedefe doğru kişileri motive etme, iletim yönü kuvvetli gibi kişisel özelliklere sahip olmalıdır. Bu özelliklerin bir kısmı doğuştan gelir veya yaşamın ilk kısmında kazanılır (Drucker, 2006).

5.4.1. Karar Verme

Karar verme, bir durum ya da problem karşısında nasıl hareket edileceğinin belirlenmesidir. Bir yöneticinin en önemli ve kritik görevi doğru kararlar almak ve bunların uygulanmasını sağlamaktır. Her karar az ya da çok risk taşır ve elbette karar verici olan yönetici, bu riskten sorumludur.

Sıradan işlerin devam edilmesinin sağlanması herkes tarafından yapılabilir. Yönetici, özellikle de yenilikçi bir yaklaşım içeren projelerin yöneticisi, projenin hedeflerine ulaşması için farklı durumlar karşısında nasıl hareket edileceğine karar vermelidir. Karar vermek, problem çözmekten farklıdır. Fayda maliyet analizi yapılarak birçok ihtimal arasından en doğru olanı bulmayı ve uygulamayı ifade eder.

6. YAZILIM PROJELERİ

Yazılım projeleri, yazılım geliştirme, mevcut bir yazılımı iyileştirme, birden çok yazılımın birbirleriyle iletişimini sağlamak ve daha birçok yenilikçi yaklaşımı içerebilir. Diğer projelerde olduğu gibi yazılım projelerinde de yönetici önemli bir oyuncudur. Doğru zamanda doğru kararı verebilmeli, insiyatif kullanabilmeli ve değişime açık olmalıdır.

6.1. Yazılım Projesi

Yazılım projesinin amacı, belirli bir hedef doğrultusunda kapsam tespitinden başlanarak, planlama, analiz, tasarım, geliştirme, gerçekleştirme, test, kurulum ve bakım aşamalarını gerçekleyerek, zaman kısıtı altında, bütçesinde ve kaliteli yazılımlar geliştirmektir.

Yazılım geliştirme, ihtiyaçların analiz edilerek bilgisayar programı haline getirilmesidir. Yazılım projesi bunu gerçekleştirmek için yapılması gerekenleri tanımlar.

Proje, yeni bir ürün, hizmet veya sonuç elde etmek için belirlenen sürede gerçekleştirilen çalışmalardır.

Yazılım Projesi: Kişi ya da kurumun belirlenmiş bir ihtiyacını analiz ederek bilgisayar ortamında karşılamak için yapılan çalışmalardır.

6.2. Yazılım Geliştirme Temel İlkeleri

Yazılım geliştirmenin belirli ilkeleri vardır. Yazılımın kalitesi bu ilkelere bağlılıkla mümkündür. Geliştirme hangi ortamda ya da teknolojiyle yapılırsa yapılsın bu ilkelere azami dikkat gösterilmelidir. Proje yöneticileri sorumlu oldukları projelerde bu ilkelerin uygulanmasını sağlamalıdırlar.

6.2.1. Basitlik

Basitlik, yazılım dünyasında geliştirilen yazılımlarda mümkün olduğunca karmaşıklıktan kaçınmak olarak ifade edilebilir. İhtiyaç analizi, tasarım ve geliştirme sırasında en basit, sade, yalın ve akılcı çözüm üretilmelidir. Yazılımın sade ve kolay ve anlaşılır olması hem sonradan yapılacak müdahaleler hem de geliştirme esnasında büyük artılar kazandıracaktır. Proje yöneticisinin bu konuya olabildiğince dikkat etmesi gereklidir. Bu ilkeyle şu kazanımlar elde edilir.

- Karmaşanın getirdiği zorluklar için para ve zaman gibi kaynaklar israf edilmez
- Tasarımın sade ve kolaylığı anlaşılabilirliğini artırır. Böylelikle kullanıcılar tarafından daha kolay kabul edilip, benimsenir.
- Kurulacak yapının basit olmasıyla planlama daha kolay yapılabilir.
- Projedeki görevler basitleşir ve çalışanların bu görevleri yapması daha kolaylaşır.

6.2.2. Yeniden Kullanılabilirlik

Yazılım geliştirme sırasında global düşünülerek kodları mümkün olduğunca yeniden kullanılabilir şekilde yazmak önemlidir. Bu şekilde bir yaklaşım hem yazılımın kalitesini artıracak hem de zaman ve maliyet tasarrufu sağlayacaktır. Günümüzde yaygın kullanılan nesneye yönelik programlama, web servis ve bileşen tabanlı mimari gibi teknolojiler yeniden kullanımı artıracak şekilde gelişmektedir.

Yeniden kullanıma olanak sağlayacak şekilde bir tasarım yapılması, yazılım geliştirmenin ilk aşamalarında biraz karmaşık ve zor olabilmektedir. Bu şekilde yapılan geliştirme ve tasarımlar kısa vadede bir kazanç sağlayamayabilir. Bu nedenle proje yöneticisi bu ilkeyi projede kullanmadan önce ayrıntılı bir fayda-maliyet analizi yapmalıdır.

6.2.3. Süreklilik

Yazılım geliştirme sonucunda ortaya çıkan ürün, kısaca yazılım olarak ifade edilen yazılım ürünüdür. Yazılım günlük sorunlara çözüm bulmak amacıyla geliştirilmemelidir. Proje yöneticisi ve proje ekibi yazılımın değişen şartlar altında çalışabilmesini sağlayacak şekilde yazılım geliştirmelidir. Aksi halde günlük problemlere çözüm sağlayan yazılım artan kullanım, veri ve kullanıcı sayısı karşısında performansını yitirebilir. Proje geliştirilirken bu uzak görüşlülük, sürekliliği sağlayan çok önemli bir etkendir.

6.2.4 İzlenebilirlik

Yazılım geliştirme süreci esnasında talep eden kullanıcıların süreci izleyebilmeleri çok önemlidir. Mümkün olduğunca yapı buna imkan sağlayabilecek şekilde basit ve anlaşılır olmalıdır. Yazılım geliştirme sadece bilim insanlarının anlayabileceği özel bir alan değil, kullanıcılara hitap eden bir mühendisliktir. (Nizam, 2014)

6.2.5. Güvenlik

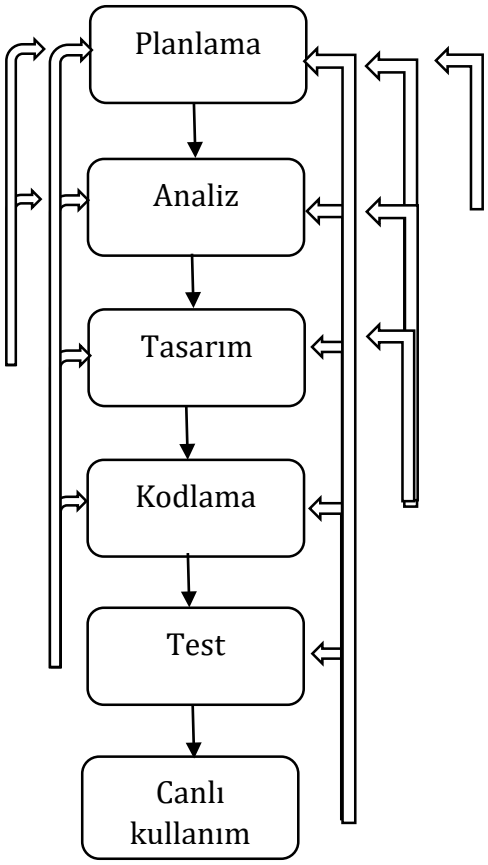
Yazılım sürece bağlı olarak çok önemli bilgileri işleyebilir. Bu nedenle bilgilerin güvenliğini sağlamak çok önemlidir. Özellikle son yıllarda bu konu yazılım dünyasında öne çıkmaktadır.

6.3. Yazılım Geliştirme Süreçleri

Yazılım bir ihtiyacın giderilmesi, bir sorunun çözümü ya da bir süreçsel iyileştirme amacından doğar. Bu amaca istinaden ilk önce yazılımın kapsamı belirlenir ve genel bir plan yapılır. Daha sonra detaylı bir şekilde ihtiyaç analizi yapılır. Analizin ortaya çıkardığı sonuçların yazılım araçlarıyla nasıl gerçekleştirileceği tasarlanır. Proje kapsamındaki tüm ekran, rapor, nesne ve veri tabanı yapıları (tablolar, tablo ilişkileri vs.) fiziksel olarak oluşturulur. Proje teste hazır olduğunda test edilir. Test başarılı geçerse eğitim ve kurulum yapılır. Tüm

bu işlemlerin sonunda en son proje canlı kullanıma alınır. Proje canlı kullanıma alındıktan sonra test aşamasında yakalanamayan hatalarla karşılaşılabilir. Aynı zamanda canlı kullanım sonrasında yeni talepler gelebilir. Bu nedenle bakım ve destek çalışması devam etmelidir.

Yazılım mühendisliği açısından geliştirme süreci: planlama, ihtiyaç analizi, tasarım, kodlama, test, devreye alma ve bakım şeklinde standartlaştırılmıştır.
Yazılım geliştirme baştan sona gerçekleştirilecek bir akış değildir. Her aşama tekrar tekrar denenerek olgunlaşır. Aşamalar arasında geriye dönüşler olabilmektedir. Yazılım geliştirmedeki en önemli sorun bu geri dönüşlerdir. Hedef bu geri dönüşleri mümkün olduğunca azaltmak hatta hiç geri dönüş yaptırtmamaktır. Bazen analizdeki belirsizlikler, müşterinin önemli bir bilgiyi son anda söylemesi, canlı kullanım öncesi gelen yeni talepler ve bazı teknik kısıtların olması geri dönüş yapılmasını zorunlu kılar. Burada önemli olan geri dönüşlerde basamaklar arası sıçrama miktarının az olmasıdır. Geriye yönelik tek adımlık sıçramaların maliyeti daha düşük olmaktadır. Ancak uzun sıçramaların, örneğin canlı geçiş sırasında fark edilen bir analiz hatasının maliyeti, bunu kodlama sırasında fark etmenin çok üzerindedir. (McConnell, 2004) Şekil 6.1'de yazılım geliştirme süreçleri gösterilmiştir.



Şekil 6.1. Yazılım geliştirme döngüsü

6.3.1. Planlama

Planlama; süre ve maliyet kısıtları altında projenin hedeflerine ulaşmak için kaynak, süre, kullanılacak teknoloji, yöntem vb. tüm ihtiyaçların belirlenmesidir. Yazılım, bir ihtiyaca istinaden ortaya çıkar. İhtiyacın ortaya çıkmasından sonra kapsam belirlenir ve planlama süreci başlar. İhtiyaç duyulan kaynakları belirlenir, tedarik edilir ve görevlendirmeler yapılır.

Projenin zamanında, bütçesinde ve beklenen kalitede yapılabilmesi için planlama çok önemlidir. Proje yöneticisi planlamayı bu şartları göz önüne alarak yapmalıdır.

6.3.2. İhtiyaç Analizi

İhtiyaç analizi, ihtiyaç ve beklentilerin tüm detaylarıyla tanımlanmasıdır. İhtiyaç aşağıdaki özellikleri taşımalıdır:

- Gerçekleştirilebilme; mevcut iş gücü ve belirlenen bütçe ile yapılabilmesi
- Doğrulanabilme; ölçülebilir ve objektif kriterlerle yapılabilmesi
- Anlaşılabilme; birden fazla anlama gelecek belirsiz ifadelerden kaçınma
- Tutarlılık; başka ihtiyaçlarla çelişkili olmamalıdır

Yazılımın tasarımını yapmadan ve geliştirmeye başlamadan önce kullanıcıların ne istediği tam olarak anlaşılmalıdır. Daha çok çözümleri değil istekleri bu aşamada belirlemek daha önemlidir. Genellikle analizin başlangıcında bu istekleri tam anlamak oldukça güçtür. Bu nedenle nasıl bir yazılım geliştirileceğini örnek model ve sunumlarla kullanıcıya gösterilmesi bu konuda yardımcı olabilir. İhtiyaç analizi yapılırken sürece dahil olan herkes karşılıklı olarak birbirine anlamalıdır. Yazılım geliştiricilerin kullanıcıların anlamaları önemli olduğu gibi, kullanıcıların da yazılımı geliştirenleri anlaması önemlidir. Bu karşılıklı empati proje üzerinde ortak mutabakatın sağlanması açısından fayda sağlayacaktır.

6.3.3. Örnek Model ve Sunum

Kullanıcılara yazılım geliştirmeye başlamadan evvel örnek bir model ve sunum üzerinden yazılımın anlatılması faydalı olacaktır. Bu anlatımda temel fonksiyonlar, ekranlar, ara yüzler vb. önemli olan kısımlar gösterilmelidir. Proje paydaşları müşteri, kullanıcı, üst yönetim veya proje ekip üyeleri bu örnek model sayesinde sistemi daha kolay anlama imkanı bulacaktır.

Ön model geliştirilirken kullanılan kodun gerçek yazılımın geliştirilmesinde kullanılmaması önerilir. (Fairley, 2009) Bu kod hızlı geliştirildiği için test ve kalite kontrol adımları atlanabilir. Dolayısıyla hedeflenen yazılım kalite standartlarına sahip olmayacaktır.

6.3.4. Tasarım

Tasarım, yapılan analiz doğrultusunda yazılım geliştirme araçlarının yetenek, kural ve kapsamları dikkate alınarak teknik olarak modellenmesi işlemidir. Tasarım ile ilişkisel veri tabanı modeli, yazılımsal nesne modeli, UML (Unified Modeling Language, Bütünleşik Modelleme Dili) şeması ve iş akış şeması gibi birçok farklı yöntem kullanılır.

İhtiyaç analizi ve tasarım uygulamada karıştırılmakta ve sık sık tek bir adımda gerçekleştirilmektedir. Ancak aralarındaki kavramsal farklılıktan dolayı bu çok hatalı sonuçlara yol açar. İhtiyaç analizinde kullanıcı isteklerinin alınmasını ve problemi öğrenmek amaçlanır. Tasarımda ise analiz sonucu ortaya çıkan ihtiyaçların, yazılım geliştirme araçlarına uygun modelleri çıkartılır. İhtiyaç analizinde amaç ihtiyacı anlamak, tasarımda ise çözüme odaklanmaktır. Proje yöneticisi analiz ve tasarım arasındaki bu ince farkı iyi bilmeli ve bu iki aşamanın birbirleriyle karışmasını ve karıştırılmasını önlemelidir.

6.3.5. Kodlama

İhtiyaç analizi çıkartılmış ve bu kapsamda tasarlanmış olan yazılım geliştirme isteği, yazılım geliştirme araçları kullanılarak bu aşamada yazılıma dönüştürülür. Seçilen yazılım diliyle kod, kod kütüphanesi, kullanıcı ara yüzü ve raporların yazımı ile veri tabanının oluşturulması bu aşamada yapılır. Kodlamadan sonra yazılım test aşamasına gönderilir. Ancak yazılım geliştiricilerin geliştirdikleri kısımlarla ilgili teste göndermeden önce birim test yapması önerilir. Bu sayede geliştirme aşamasında bazı sorunlar yakalanabilir.

6.3.6. Test

Test, bu aşamaya gelinceye kadar tüm yapılanların kontrol edilmesidir. Analiz için istenenlerin doğru anlaşılıp anlaşılmadığı test edilir, tasarım ve kodlamada

ise yapılanların doğru çalışıp çalışmadığı test edilir. Test sonuçlarına göre yazılımın tasarım ve kodunda ve hatta analizinde değişikliğe gidilebilir.

Test yapılırken analiz ve tasarım dikkatlice incelenip ona göre yazılım test edilmelidir. Eksik bilgi ile deneme yanılma yapılması doğru değildir. Deneme yanılma yazılım geliştirirken tercih edilecek bir yöntem olmamalıdır. (McConnell, 2004)

6.3.7. Canlı Geçiş

Test aşaması da tamamlandıktan sonra yazılım artık canlı kullanıma alınabilir. Canlı kullanıma alınmadan önce kullanıcı eğitimlerinin tamamlanması, alındıktan sonra ise destek çalışmalarının planlanması gereklidir. Geliştirme ve testi tamamlanmış yazılımın canlı kullanıma alınması kolay gibi gözükse de oldukça zor ve detaylı planlanması gereken bir yazılım geliştirme sürecidir. Her ne kadar geliştirme ve test sırasında yazılım sürekli kontrol edilse de projenin canlı kullanıma geçmesiyle birlikte bu aşamalarda gözden kaçan hatalarla karşılaşılabilir. Ayrıca artan kullanıcı veri sayısı performans sorunu oluşturabilir. Yazılım canlı kullanıma alındıktan sonra belirli bir süre gözlemlenmeli çıkabilecek sorunlara anında müdahale edilmelidir.

6.4. Yazılım Süreç Modelleri

Yazılım geliştirme sırasında izlenecek farklı yollar, yazılım süreç modelleri olarak standartlaşmıştır. Süreç modeli ile yazılım geliştirme aşamalarının sırası ve aralarındaki geçiş kriterleri belirlenir. (Boehm B. , 1989) Bir süreç modelini izleyerek yazılım geliştirmek, tüm yazılım sürecini bir bütün olarak görmeyi ve gerekli aşamaların hatırlanmasını sağlar. Böylelikle geliştirme esnasında önemli bir işlemin atlanmasına engel olur.

Yazılımın analizden tasarıma sürekli ilerleyen bir şekilde geliştirilmesi, doğrusal bir model olarak adlandırılır. Doğrusal model şelale ve V modelinden oluşur. Yazılım geliştirilirken ihtiyaç ve tasarım kararlarının değiştirilmesi birçok geri

dönüşe yol açar. Bu sorunları yönetebilmek için yinelemeli modeller ortaya çıkmıştır. Artımlı, evrimsel ve sarmal model buna örnektir.

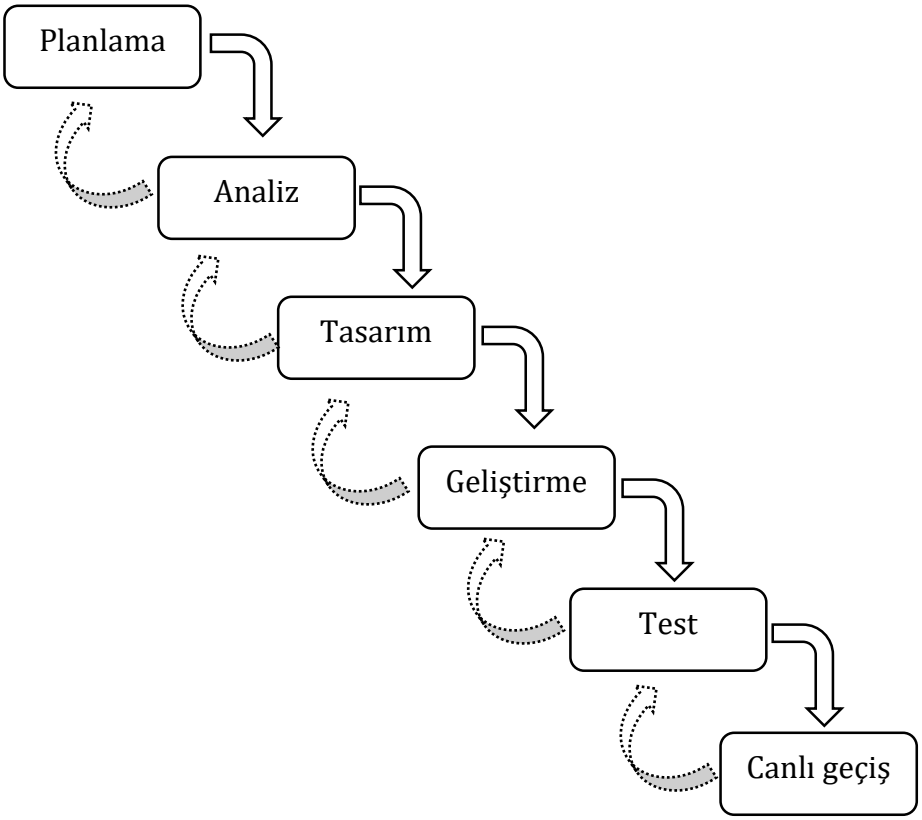
6.5. Doğrusal Modeller

Doğrusal modeller, yazılım geliştirme sürecinde analizden canlı geçişe kadar ki aşamaların, birbirini art arda takip ettiği ve geri dönüşlerin az yaşandığı varsayılarak kullanılan modellerdir. Bir aşamadan diğerine geçmek için ilgili aşamadaki tüm işlerin tamamlanmış olması gerekmektedir. Yazılım geliştirmede sık kullanılan bir modeldir. Ancak günümüzde teknik alt yapıların sürekli değişmesi yüzünden doğrusal modeller önemli dezavantaja sahiptir.

6.5.1. Şelale Modeli

Doğrusal bir model olan şelale modelinde yazılım geliştirme aşamalarının analizden canlı geçişe kadar doğrusal olarak ilerlediği varsayılır. Analizde kullanıcı ihtiyaçlarının tamamen anlaşıldığı aynı şekilde tasarımda analizin eksiksiz olduğu kabul edilerek bir sonraki adıma geçilir. (Winston, 1970)

Şelale modeli, ilk önerilen modellerden biridir. Bu nedenle yaygın kullanımı vardır. Bu modelin en önemli dezavantajı, kullanıcı ihtiyaç ve beklentilerinin ilk anda tamamen analiz edilememesidir. Şekil 6.2'de şelale modeli gösterilmiştir.



Şekil 6.2. Şelale Modeli

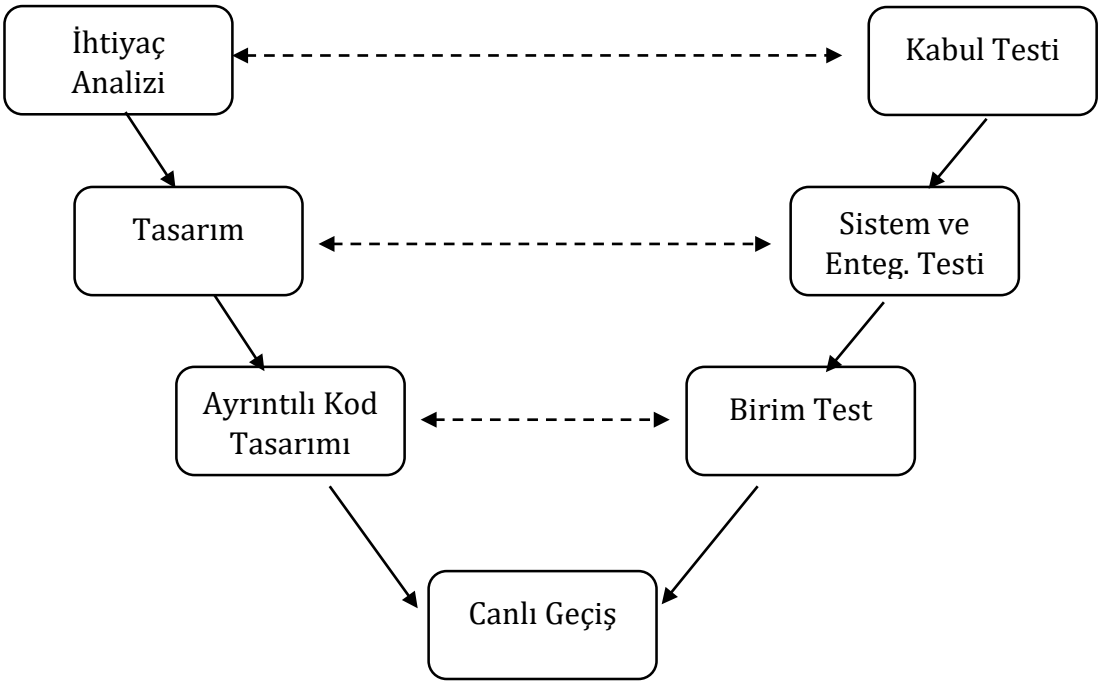
Şelale modeli bazı araştırmalarda geri beslemeleri de olan bir geliştirme süreci olarak tanıtılmıştır. Ancak buna rağmen çalışmalarda doğrusal kabul edilmiştir. (Pressman, 2009)

Şelale modelinde her aşama sonunda kontrol planlanmalıdır. Kontrol sırasında herhangi bir sorunla karşılaşılmaz ise bir sonraki aşamaya geçilir. Aksi takdirde ilgili aşama yeniden düzenlenir. Bu modelde ihtiyaç analizinin kapsamlı ve eksiksiz yapılması ve kullanıcılara hazırlanan örnek modellerin gösterilmesi modelin sorunsuz çalışmasına büyük katkı sağlayacaktır.

6.5.2. V Modeli

V modeli, şelale modelinin özellikle kontrol aşamalarının daha organize edilmiş hali olarak görülebilir. Her aşama kendi kontrol aşamasıyla eşleştirilerek ”V” harfine benzer şekille gösterildiği için bu isimle anılır. Bu modelle analiz ile kabul testi, tasarım ile sistem ve entegrasyon testi, kodlama ile birim testi eşleştirilir.

V modelinde yapılan çalışmaların testi ve doğrulaması planlı bir şekilde yapılır. Böylece hataların daha kolay fark edilmesi ve düzeltilmesi sağlanır. (Pressman, 2009) Şekil 6.3'de V modeli detayları ile gösterilmiştir.



Şekil 6.3. V Modeli

6.6. Yinelemeli Geliştirme

Yinelemeli (iterasyonlu) geliştirme yazılımı bir seferde geliştirerek değil, gerekli görülen fazlara bölerek aşama aşama geliştirmektir. Öncelikle ihtiyaç analizine göre temel bir sürüm geliştirmek sonra bu sürüme yeni özellikler ekleyerek kullanıcıların ihtiyacını karşılayacak en son sürüme ulaşmak yinelemeli geliştirmeyi ana hedefidir. (Bittner ve Spence, 2006)

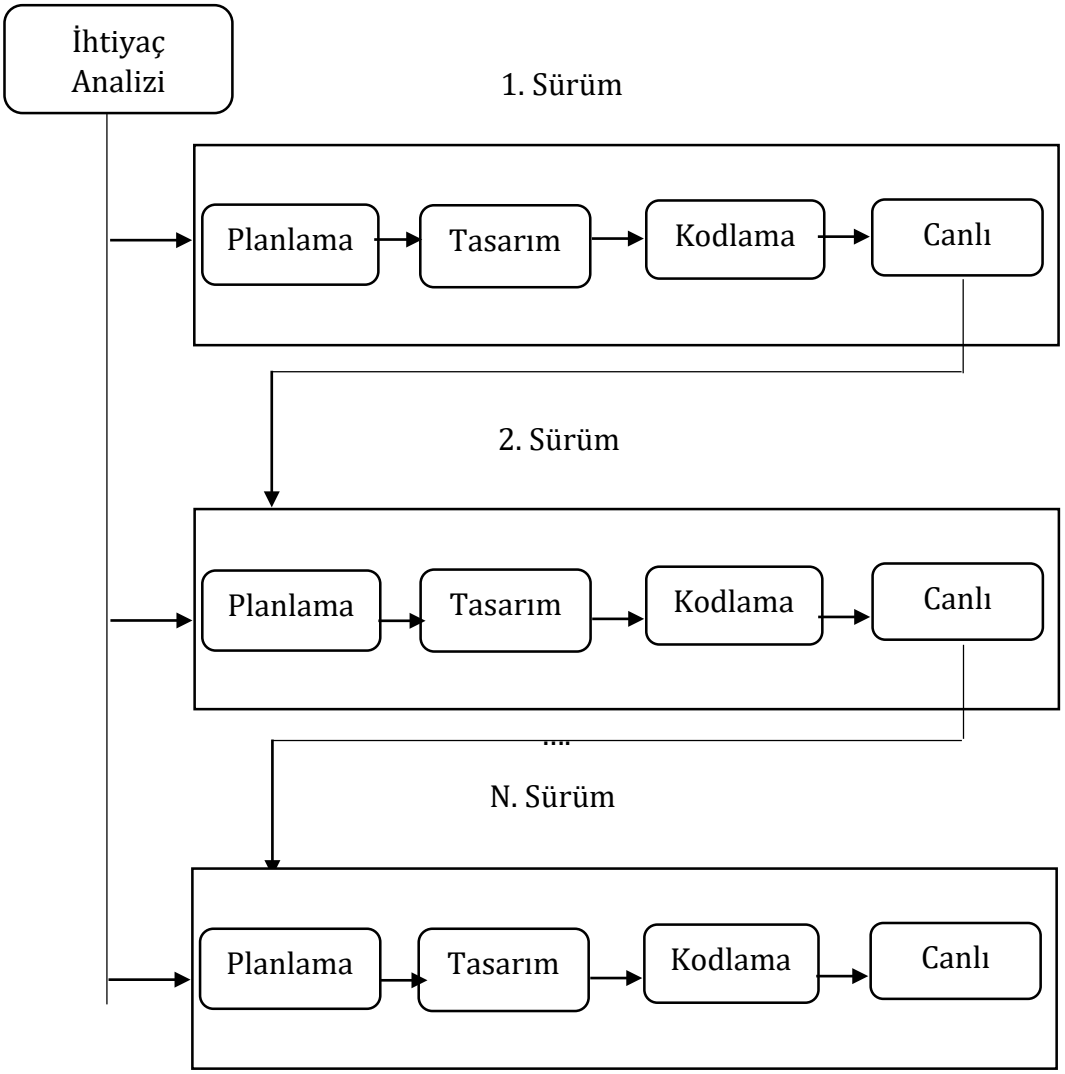
Yinelemeli modelde yazılım daha erken canlı kullanıma alınabilir. Sürümler adım adım test edilerek olgunlaşacağından hatanın tüm sisteme etkisi büyük ölçüde engellenir. Yazılım geliştirmede sıkça karşılaşılan proje aşamasında yeni kullanıcı talepleri bu modelde temel sürümde yer almaz. Projenin diğer fazlarına ötelenir. Bu şekilde proje başlangıcında belirlenen ilk ihtiyaç analizi ve tasarıma bağlı kalınarak ilk sürüm yayınlanabilir. Bazı durumlarda yazılımın kısa sürede canlı

kullanıma alınması istenebilir. Beklenen sürede yazılımın tümünün bitirilemeyeceği ön görülüyorsa, proje fazlara bölünür. Fazlara bölünerek yapılan geliştirme ile temel ve elzem beklentiler kısa sürede kullanıcıya sunulur.

6.6.1. Artımlı Geliştirme Modeli

Artımlı geliştirme; ihtiyaç analizi büyük ölçüde belli olduktan sonra yazılım geliştirme sürecindeki işlemlerin fazlara bölünerek yapılmasıdır. Her faz sonucunda bir sürüm yayınlanır. Bu sürümlerde ihtiyaçların belirli bir kısmı karşılanır. Her yeni sürümle yazılım eklenen yeni özelliklerle daha gelişmiş bir hale gelir.

Bazı sürümler kullanıcıların beklentilerini karşılarken bazı sürümler üst yönetimin beklentilerini karşılar. Baz durumlarda proje ekibinin tasarrufunda proje ekibine yönelik sürümler de yayınlanabilir. Sürümlerin özellikleri beklentiler göz önüne alınarak belirlenir. (Boehm B. W., 1988) Şekil 6.4'te artımlı geliştirme modeli gösterilmiştir.



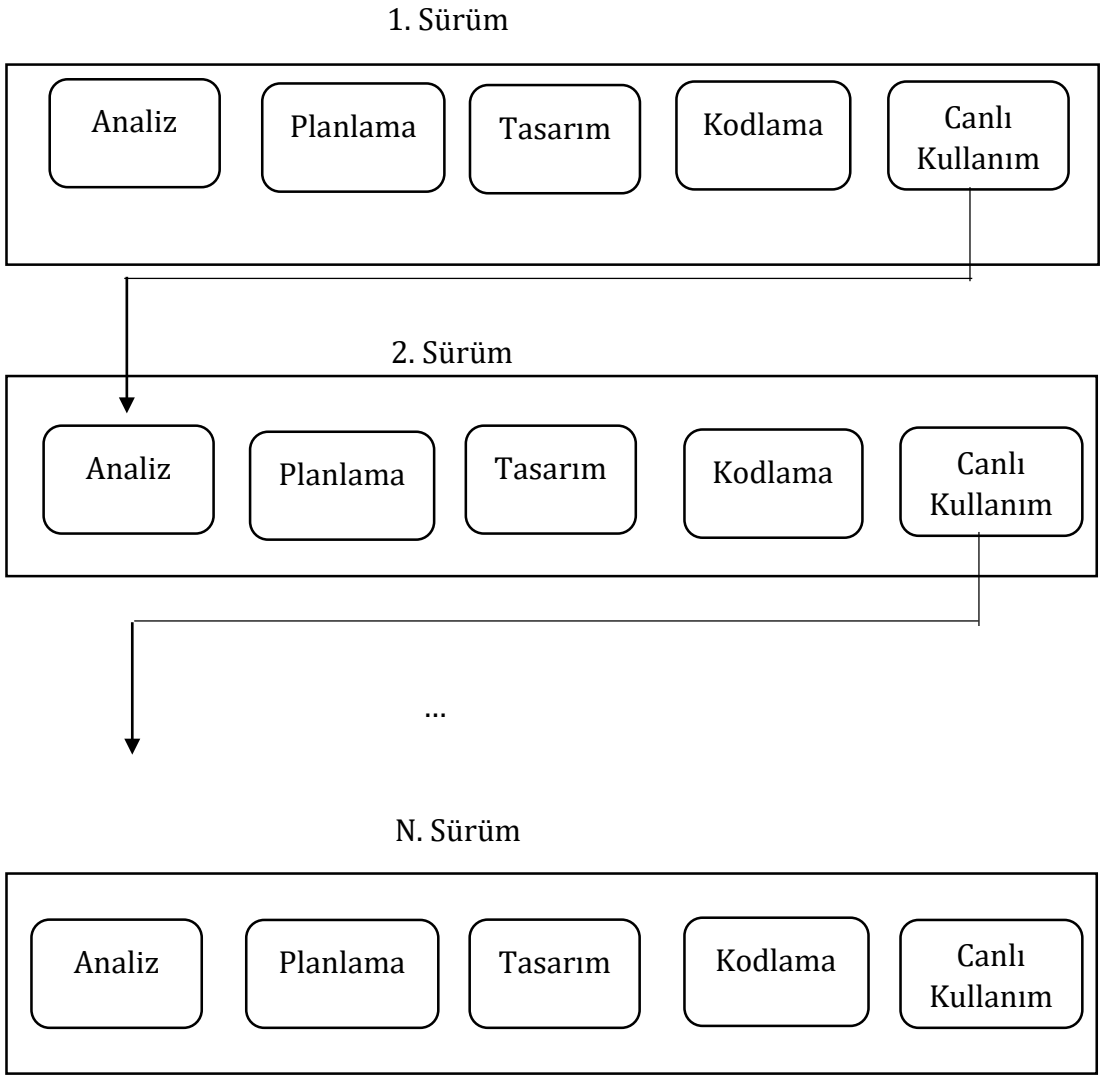
Şekil 6.4. Artımlı Geliştirme Modeli

6.6.2. Evrimsel Geliştirme

Evrimsel geliştirme, ihtiyaç analizinden başlar. Sonrasında doğrusal modeldeki gibi geliştirmeden teste kadar tüm aşamaları içeren çalışmalar bütünü oluşturur. Yine artımlı modelde olduğu gibi süreç fazlara bölünerek yönetilir. Her faz önceki fazlarla bağlantılıdır. Bir nevi fazlar arası küçük bir şelale modeli oluşur. (Bittner ve Spence, 2006)

Artımlı modelde ihtiyaçlar en başta kapsamlı olarak analiz edilir ve büyük ölçüde sınırları belirlenir. Daha sonraki fazlarda bu temel analiz çok fazla değişikliğe uğramadan yeni ihtiyaçlar karşılanır. Evrimsel geliştirmede ise her fazda yeniden

bir ihtiyaç analizi yapılır. Projenin her aşamasında kapsamlı değişiklikler olabilir. Evrimsel geliştirme özellikle ihtiyaçların ilk aşamada belirlenemediği ya da sürekli değiştiği durumlara çözüm bulabilmek için geliştirilmiş bir geliştirme yöntemidir.

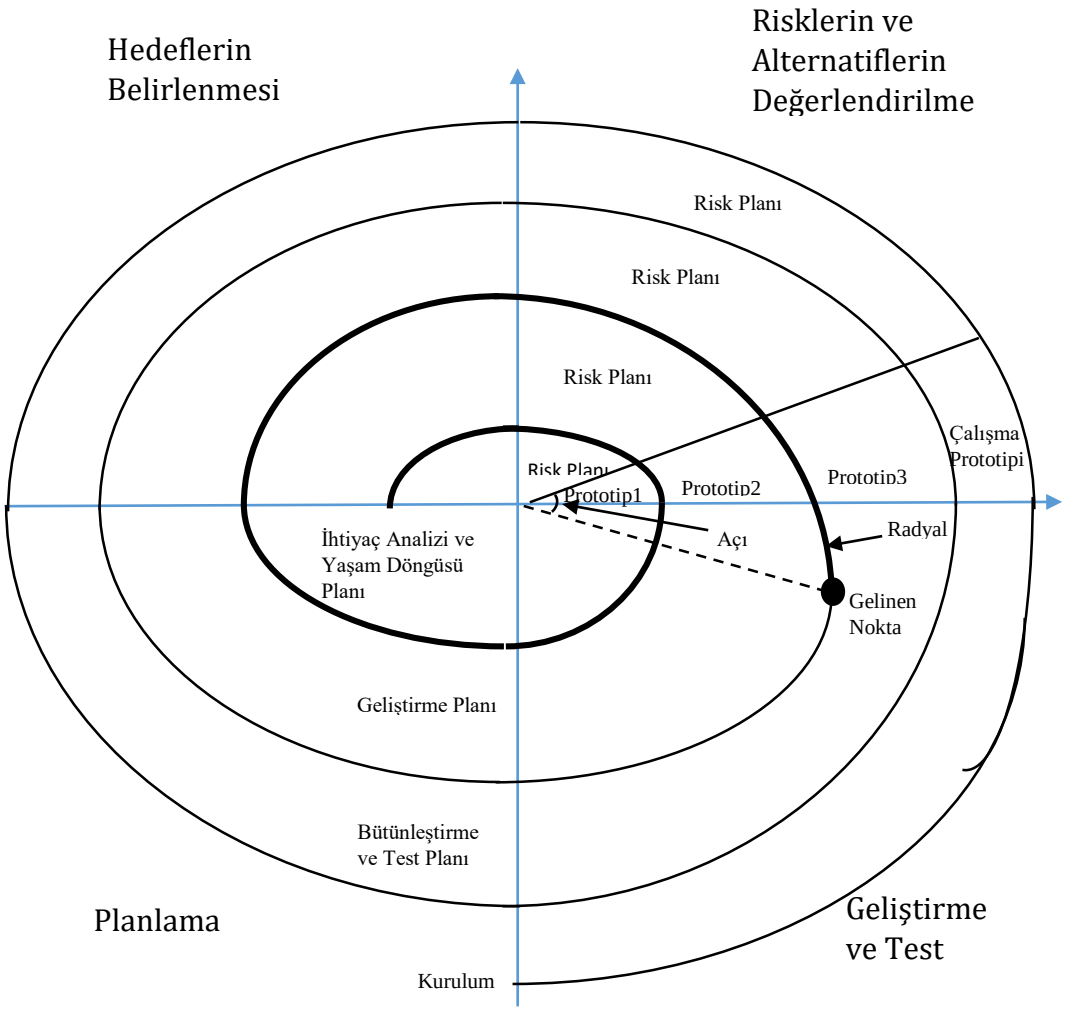


Şekil 6.5. Evrimsel Yazılım Geliştirme Modeli

6.6.3. Sarmal Model

Sarmal model 1988 yılında Boehm tarafından önerilmiştir. Sarmal modeli, yinelemeli geliştirme ve şelale modelinin üstünlüklerinin risk temelli bir yaklaşımla ele alarak bir bütün olarak ifade etmeyi amaçlamaktadır. (Boehm B. W., 1988) Bu model ile sistem daha net ve anlaşılır bir hale gelir. Gerçekleşen

kısım büyüdükçe risk azalır. Tanımlanan kilometre taşları vasıtasıyla geliştirilen yazılımın beklenen ihtiyaçlarla uyuştuğu konusunda kullanıcılarla mutabakat sağlanır. Şekil 6.6'da sarmal model detaylı olarak gösterilmiştir.



Şekil 6.6. Sarmal Model

Her sürüm bir dairesel çevrimdir ve geliştirilmesi şelale modeli şeklindedir. İlk sürümler daha çok prototip ve kullanıcı ihtiyaçlarını anlamayı hedeflerken sonraki sürümler geliştirilmesi düşünülen uygulamaya yakın özel fonksiyonlara sahip olmaya başlar. Modelde her sarmal, yaklaşık eşit gösterilmekle birlikte, projede yapılması planlananlara göre gerçek hayatta bazı sarmallar daha dar, daha geniş veya düzgün olmayan şekillerde olabilir.

Sarmal modelde sistem bir seri evrimsel sürüm şeklinde geliştirilir. Daire yayının uzunluğuna dayalı (radyal) uzaklık o tarihe kadarki bitirilen işler için harcanan toplam maliyeti, açısal uzaklık sarmalın her dönüşünü tamamlamayla sağlanan ilerlemeyi gösterir. (Nizam, 2014) Şekil 6.6 'da incelenen proje, geldiği noktada geliştirme ve test aşamasındadır. Bu noktaya harcanan toplam maliyet sarmal üzerindeki kalın çizgi ile gösterilmiştir.

7. ÖRNEK PROBLEM

7.1. Doğrusal Optimizasyon Problemi

Bir yazılım şirketi bir projede 10 adet çalışanını görevlendirecektir. Proje sürecinde yapılacak çalışmalar; analiz, kodlama, test ve canlı geçiş olarak planlanmaktadır. Bunun için çalışanlar görev tanımları ve yetkinliklerine göre iki ekip olarak sınıflandırılmıştır. Yazılım ekibi; y_1, y_2, y_3, y_4 çalışanlarından oluşmakta ve yalnızca yazılım, test ve canlı geçiş aşamalarında görev alabilmektedirler. Analiz ekibi ise; a_1, a_2, a_3, a_4, a_5, a_6 çalışanlarından oluşmakta ve yalnızca analiz, test ve canlı geçiş aşamalarında görev alabilmektedirler.

Bu çalışanların projede görev alabilecekleri aşamalar, yaklaşık olarak çalışma süreleri ve birim maliyetleri çizelge 7.1 'de verilmiştir.

Çizelge 7.1 Çalışma süresi ve birim maliyet çizelgesi

Çalışan	İş (Saat)				Saat Ücreti (TL)
	Analiz	Kodlama	Test	Canlı Geçiş	
Y_1	-	84	78	66	38
Y_2	-	82	80	64	28
Y_3	-	102	78	68	24
Y_4	-	106	76	68	26
A_1	70	-	66	54	28
A_2	94	-	52	56	24
A_3	78	-	58	62	38
A_4	86	-	70	60	36
A_5	112	-	64	52	20
A_6	102	-	72	50	22

Kısıtlar şu şekilde belirlenmiştir.

- Her bir yazılımcının bu projede en az 8 saat, her bir analistin ise bu projede en az 16 saat çalışması istenmektedir.
- Verimli bir proje olması için kodlamaya en fazla 86 saat, analize 100 saat, teste 80 saat, canlı geçişe 64 saat ayrılmalıdır.

- Belirlenen kalite standartları çerçevesinde projede kodlamaya en az 82 saat, analize 80 saat, teste 56 saat, canlı geçişe 60 saat ayrılmalıdır.

Çalışanların işleri yapabilme yetenekleri ve maliyetleri, aynı zamanda belirtilen kısıtlar da dikkate alarak bu projenin minimum maliyetle gerçekleşmesini sağlayalım.

7.2. Problemin Çözümü

Doğrusal bir optimizasyon problemi olarak tanımladığımız problemin çözümü için, ilk önce her bir çalışanın belirtilen analiz, tasarım, kodlama, test ve canlı geçiş işlerinin yapılma maliyetlerini çizelge 7.2 'de gösterelim.

Çizelge 7.2. Maliyet çizelgesi

Çalışan	Analiz	Kodlama	Test	Canlı Geçiş
Y_1	-	3192	2964	2508
Y_2	-	2296	2240	1792
Y_3	-	2448	1872	1632
Y_4	-	2756	1976	1768
A_1	1960	-	1848	1512
A_2	2256	-	1248	1344
A_3	2964	-	2204	2356
A_4	3096	-	2520	2160
A_5	2240	-	1280	1040
A_6	2244	-	1584	1100

Değişkenler çizelge 7.3'de belirlenmiştir.

Çizelge 7.3 Değişken çizelgesi

Çalışan	Analiz	Kodlama	Test	Canlı Geçiş
Y_1	-	X_1	X_2	X_3
Y_2	-	X_4	X_5	X_6
Y_3	-	X_7	X_8	X_9
Y_4	-	X_{10}	X_{11}	X_{12}
A_1	X_{13}	-	X_{14}	X_{15}
A_2	X_{16}	-	X_{17}	X_{18}
A_3	X_{19}	-	X_{20}	X_{21}
A_4	X_{22}	-	X_{23}	X_{24}
A_5	X_{25}	-	X_{26}	X_{27}
A_6	X_{28}	-	X_{29}	X_{30}

En az maliyet için amaç fonksiyonunu yazalım.

$$\text{Min } C = 3192 X_1 + 2964 X_2 + 2508 X_3 + 2296 X_4 + 2240 X_5 + 1792 X_6 + 2448 X_7 + 1872 X_8 + 1632 X_9 + 2756 X_{10} + 1976 X_{11} + 1768 X_{12} + 1960 X_{13} + 1848 X_{14} + 1512 X_{15} + 2256 X_{16} + 1248 X_{17} + 1344 X_{18} + 2964 X_{19} + 2204 X_{20} + 2356 X_{21} + 3096 X_{22} + 2520 X_{23} + 2160 X_{24} + 2240 X_{25} + 1280 X_{26} + 1040 X_{27} + 2244 X_{28} + 1584 X_{29} + 1100 X_{30}$$

Kısıtlar:

Çalışma kısıtları: Her bir yazılımcı bu projede en az 8 saat, her bir analist ise en az 16 saat çalışmalı.

1. Yazılımcı $X_1 + X_2 + X_3 >= 8$

2. Yazılımcı $X_4 + X_5 + X_6 >= 8$

3. Yazılımcı $X_7 + X_8 + X_9 >= 8$

4. Yazılımcı $X_{10} + X_{11} + X_{12} >= 8$

1. Analist $X_{13} + X_{14} + X_{15} >= 16$

2. Analist $X_{16} + X_{17} + X_{18} >= 16$

3. Analist $X_{19} + X_{20} + X_{21} >= 16$

4. Analist $X_{22} + X_{23} + X_{24} >= 16$

5. Analist $X_{25} + X_{26} + X_{27} >= 16$

6. Analist $X_{28} + X_{29} + X_{30} >= 16$

İşleri en fazla sürede bitirme kısıtları:

$X_1 + X_4 + X_7 + X_{10} <= 86$ (Kodlama en fazla 86 saatte bitmeli)

$X_{13} + X_{16} + X_{19} + X_{22} + X_{25} + X_{28} <= 100$ (Analiz en fazla 100 saat)

$X_2 + X_5 + X_8 + X_{11} + X_{14} + X_{17} + X_{20} + X_{23} + X_{26} + X_{29} <= 80$ (Test en fazla 80 saat)

$X_3 + X_6 + X_9 + X_{12} + X_{15} + X_{18} + X_{21} + X_{24} + X_{27} + X_{30} <= 64$ (Canlı geçiş en fazla 64 saat)

İşleri en az sürede bitirme kısıtları:

$X_1 + X_4 + X_7 + X_{10} >= 82$ (Kodlamaya en az 82 saat ayrılmalı.)

$X_{13} + X_{16} + X_{19} + X_{22} + X_{25} + X_{28} >= 80$ (Analiz en az 80 saat)

$X_2 + X_5 + X_8 + X_{11} + X_{14} + X_{17} + X_{20} + X_{23} + X_{26} + X_{29} >= 56$ (Test en az 56 saat)

$X_3 + X_6 + X_9 + X_{12} + X_{15} + X_{18} + X_{21} + X_{24} + X_{27} + X_{30} >= 60$ (Canlı geçiş en az 60 saat)

Bu projeyi matematiksel olarak ifade ettiğimizde, 30 değişkenli 18 kısıtı olan bir problem elde etmiş oluruz.

7.2.1. Doğrusal programlama ile çözüm

Bu optimizasyon probleminin doğrusal programlama ile çözümü sağlanabilir. Doğrusal programlama ile çözümü için matlab kullanılmış ve aşağıdaki sonuçlar elde edilmiştir. Ek A'da doğrusal programlama için yazılan matlab kodları bulunmaktadır.

1. Yazılımcı $X_1 = 8, X_2 = 0, X_3 = 0$

2. Yazılımcı $X_4 = 58, X_5 = 0, X_6 = 0$

3. Yazılımcı $X_7 = 8, X_8 = 0, X_9 = 0$

4. Yazılımcı $X_{10} = 8, X_{11} = 0, X_{12} = 0$

1. Analist $X_{13} = 80, X_{14} = 0, X_{15} = 0$

2. Analist $X_{16} = 0, X_{17} = 40, X_{18} = 0$

3. Analist $X_{19} = 0, X_{20} = 16, X_{21} = 0$

4. Analist $X_{22} = 0, X_{23} = 0, X_{24} = 16$

5. Analist $X_{25} = 0, X_{26} = 0, X_{27} = 28$

6. Analist $X_{28} = 0, X_{29} = 0, X_{30} = 16$ saat olarak bulunur.

Proje maliyeti ise;
Min C = 523.600 TL olarak bulunur.

7.2.2. Genetik Algoritma Kullanılarak Çözüm

Aynı problemin matlab yardımıyla Genetik Algoritma ile çözümünü sağlayalım. Genetik Algoritma çalışma özelliklerini şu şekilde seçelim. Üreme fonksiyonu olarak lineer uygulanabilen @gacreationlinearfeasible fonksiyonunu, çaprazlama için "iki nokta" çaprazlama yöntemini, seçim işlemi olarak turnuva yöntemini (bu örnekte turnuva boyutu 4 olarak belirlenmiştir), hibrid kısmında ise patternsearch seçeneğini seçelim.

```
%üreme fonksiyonu olarak lineer uygulanabilir fonksiyonu seçilmiş
options = gaoptimset('CreationFcn',@gacreationlinearfeasible)

%çaprazlama için "iki nokta" çaprazlama seçilmiştir.
options = gaoptimset(options,'CrossoverFcn',@crossovertwopoint)

%seçim işlemi olarak turnuva yöntemi seçilir.
%Turnuva boyutu en az 2 olmalıdır.
options =
gaoptimset(options,'SelectionFcn',{@selectiontournament,4})
```

```
%hibrid kısmında pattern search seçilmiştir
options = gaoptimset(options,'HybridFcn',@patternsearch)
```

Amaç fonksiyonu simple_fitness m, kısıtlar ise simple constraint m dosyasında Ek B'deki gibi tanımlanmıştır. Genetik algoritma çözümü için yazılan matlab kodları Ek C'dedir. Program farklı zamanlarda 6 kere çalıştırılmış ve çizelge 7.4'teki sonuçlar elde edilmiştir.

Çizelge 7.4 Genetik algoritma farklı çalışma sonuçları

	1. Çlş	2. Çlş	3. Çlş	4. Çlş	5. Çlş	6. Çlş
Y1 ($X_1 + X_2 + X_3$)	8	8	8	8	8	8
Y2 ($X_4 + X_5 + X_6$)	75,55	70,68	67,51	66,7	69,54	68,2
Y3 ($X_7 + X_8 + X_9$)	8	8	8	8	8	8
Y4 ($X_{10} + X_{11} + X_{12}$)	8	8	8	8	8	8
A1 ($X_{13} + X_{14} + X_{15}$)	71,96	64,99	61,67	79,44	68,19	77,64
A2 ($X_{16} + X_{17} + X_{18}$)	42,45	54,33	60,83	35,13	41,91	43,98
A3 ($X_{19} + X_{20} + X_{21}$)	16	16	16	16	16	16
A4 ($X_{22} + X_{23} + X_{24}$)	16	16	16	16	16	16
A5 ($X_{25} + X_{26} + X_{27}$)	16	16	16	21,12	26,36	16
A6 ($X_{28} + X_{29} + X_{30}$)	16	16	16	19,60	16	16,17
Kodlama ($X_1 + X_4 + X_7 + X_{10}$)	82	82	82	82	82	82
Analiz ($X_{13} + X_{16} + X_{19} + X_{22} + X_{25} + X_{28}$)	80	80	80	80	80	80
Test ($X_2 + X_5 + X_8 + X_{11} + X_{14} + X_{17} + X_{20} + X_{23} + X_{26} + X_{29}$)	56	56	56	56	56	56
Canlı ($X_3 + X_6 + X_9 + X_{12} + X_{15} + X_{18} + X_{21} + X_{24} + X_{27} + X_{30}$)	60	60	60	60	60	60
Proje maliyeti	542.926 tl	540.609 tl	545.302 tl	528.813 tl	539.862 tl	533.919 tl
Çözüm zamanı	3,4592 sn	2,9549 sn	2,6948 sn	3,7664 sn	1,9797 sn	2,0292 sn

Son çalıştırmanın detayları aşağıda verilmiştir.

1. Yazılımcı $X_1 = 5,9$ / $X_2 = 0$ / $X_3 = 2,1$

2. Yazılımcı $X_4 = 68,2$ / $X_5 = 0$ / $X_6 = 0$

3. Yazılımcı $X_7 = 7,9$ / $X_8 = 0,1$ / $X_9 = 0$

4. Yazılımcı $X_{10} = 0$ / $X_{11} = 3,32$ / $X_{12} = 4,68$

1. Analist $X_{13} = 77,64$ / $X_{14} = 0$ / $X_{15} = 0$

2. Analist $X_{16} = 0$ / $X_{17} = 27,76$ / $X_{18} = 16,22$

3. Analist $X_{19} = 0,05$ / $X_{20} = 15,95$ / $X_{21} = 0$

4. Analist $X_{22} = 0,2$ / $X_{23} = 0$ / $X_{24} = 15,80$

5. Analist $X_{25} = 2,04$ / $X_{26} = 4,31$ / $X_{27} = 9,65$

6. Analist $X_{28} = 0,07$ / $X_{29} = 4,55$ / $X_{30} = 11,55$ saat olarak bulunur.

7.3. Bulgular

Örnek bir yazılım projesi için tasarlanan minimum amaç fonksiyonlu optimizasyon problemi, doğrusal programlama ve genetik algoritma ile ayrı ayrı çözülmüştür. Doğrusal programlama ve Genetik algoritma ile çözümlenen, belirlenen işlerde kullanılan kaynak ve çalışma süreleri çizelge 7.5'de gösterilmiştir. Doğrusal programlama ile maliyet 523.600 TL bulunmuştur. Sezgisel yaklaşım olan Genetik algoritma ile problem 6 defa çözümlenmiş ve maliyet ortalama 538.571 TL olarak hesaplanmıştır. Çizelge 7.4'te görüldüğü gibi farklı zamanlarda çalıştırılan genetik algoritma çözümlerinde birbirine yakın değerler elde edilmiştir. Bu sonuçların ortalaması Çizelge 7.5'de doğrusal programlamanın sonuçlarıyla karşılaştırılmış yine birbirine yakın değerler elde edilmiştir.

Çizelge 7.5 Doğrusal Programlama ve Genetik Algoritma Çözüm sonuçları

Çalışan ve iş adımları	Çalışma süresi kısıtları	Doğrusal Programlama	Genetik Algoritma
		Çalışma süresi	Çalışma süresi
Yazılımcı 1	$X_1 + X_2 + X_3 >= 8$	8	8
Yazılımcı 2	$X_4 + X_5 + X_6 >= 8$	58	69,69
Yazılımcı 3	$X_7 + X_8 + X_9 >= 8$	8	8
Yazılımcı 4	$X_{10} + X_{11} + X_{12} >= 8$	8	8
Analist 1	$X_{13} + X_{14} + X_{15} >= 16$	80	70,64
Analist 2	$X_{16} + X_{17} + X_{18} >= 16$	40	46,43
Analist 3	$X_{19} + X_{20} + X_{21} >= 16$	16	16
Analist 4	$X_{22} + X_{23} + X_{24} >= 16$	16	16
Analist 5	$X_{25} + X_{26} + X_{27} >= 16$	28	18,58
Analist 6	$X_{28} + X_{29} + X_{30} >= 16$	16	16,62
Kodlama	$82 <= X_1 + X_4 + X_7 + X_{10} <= 86$	82	82
Analiz	$80 <= X_{13} + X_{16} + X_{19} + X_{22} + X_{25} + X_{28} <= 100$	80	80
Test	$56 <= X_2 + X_5 + X_8 + X_{11} + X_{14} + X_{17} + X_{20} + X_{23} + X_{26} + X_{29} <= 80$	56	56
Canlı	$60 <= X_3 + X_6 + X_9 + X_{12} + X_{15} + X_{18} + X_{21} + X_{24} + X_{27} + X_{30} <= 64$	60	60
Proje maliyeti		523.600 tl	538.571 tl
Çözüm zamanı		0,3019 sn	2,8139 sn

7.4. Doğrusal Olmayan Optimizasyon Problemi

Proje minimum maliyetle devreye alınmış ve kullanılmaktadır. Projeyle hayata geçirilen ve kullanılan uygulamalar kalite kontrol standartları gereği belirli oranda hata ile çalışabilir. Hata oranı belirlenen tolerans değerlerinin üzerine çıktığında yeni sürüm çıkartılacaktır.

Kullanım kaynaklı hatalar X_2 ile gösterilmiş, kullanım miktarına göre (adım sayısı) uygulamalarda alınan hata miktarı ise $2X_1^{0,5} + X_2$ olarak ifade edilmiştir.

Yeni sürüm yayınlanmadan projenin en yüksek kullanıcı sayısı ile çalışması istenmektedir.

Adım (step) sayısı ile kullanım kaynaklı hata sayısının kullanıcı sayısı ile arasındaki ilişki aşağıdaki fonksiyon ile belirtilmiştir.

$F(X) = X_1^{0,5} + 2X_2^{0,5}$

Toplam hata sayısı: $2X_1^{0,5} + X_2$

Kullanım kaynaklı hata sayısı: X_2

Uygulamadan kaynaklı hata sayısı: $2X_1^{0,5}$ olarak belirlenmiştir.

Problemdeki kısıtlar şu şekilde belirlenmiştir:

Proje sonucunda devreye alınan tüm uygulamalardaki hatalar 20'den büyük olduğunda yeni sürüm çıkartılacaktır.

$2X_1^{0,5} + X_2 <= 20$

Kullanıcı eğitimleri ve diğer yapılacak çalışmalar ile kullanım kaynaklı hata sayısının 7'den büyük olmayacağı beklenmektedir.

$X_2 <= 7$

Hataların tümü kullanım kaynaklı olamaz.

$X_1 >= 0, X_2 >= 0$

7.4.1. Doğrusal Olmayan Problem İçin Fmincon Fonksiyonu ile Çözüm

Amaç fonksiyonu ve kısıtları tanımlanan problem için matlab'da bulunan fmincon fonksiyonu kullanılarak çözüm bulunabilir. Fmincon fonksiyonu kısıtlı, doğrusal

olmayan ve çok değişkenli problemlerin çözümü için kullanılır. Minimum amaçlı fonksiyonlar için tasarlanmış bir yöntemdir. Problem maximum amaçlı bir problem olduğundan amaç fonksiyonu -1 ile çarpılarak kullanılabilir.

Amaç fonksiyonu nl_func m, kısıtlar ise nl_cons m dosyasında Ek D'deki gibi tanımlanmıştır. Fmincon fonksiyonu ile çözüm için yazılılan matlab kodları Ek E'dedir.

Çözüme ilişkin sonuçlar aşağıda gösterilmiştir.

X_1 = 64, X_2 =4 bulunmuş. Bu değerlere göre en iyi F(X) değeri 12 olarak bulunmuştur. Buna göre yeni versiyona gerek kalmadan, proje kapsamındaki uygulamaların 12 kullanıya hizmet edebileceği görülmüştür. Fmincon ile 0,0286 sn. 'de çözüme ulaşılmıştır.

7.4.2. Doğrusal Olmayan Problem İçin Genetik Algoritma Çözümü

Fmincon fonksiyonuna girerek çözümü tespit edilen doğrusal olmayan ve literatürde çözümü zor olarak nitelendirilen problemi, matlab yardımıyla Genetik Algoritma ile çözümünü sağlanacaktır. Bunun için ga fonksiyonu kullanılacak ve genetik algoritma özellikleri gaoptimset parametresinde tanımlanacaktır.

Amaç fonksiyonu nl_func m, kısıtlar ise nl_cons m dosyasında Ek D'deki gibi tanımlanmıştır. Genetik algoritma çözümü için yazılan matlab kodları Ek F'dedir.

Çözüm için populasyon büyüklüğü 50, 100 ve 150 belirlenerek çalıştırıldığında, nesillere ait hesaplamalar çizelge 7.6, çizelge 7.7, çizelge 7.8'deki gibi bulunmuştur.

Çizelge 7.6 Nesillere ait hesaplamalar (Populasyon 50)

Çalıştırma sırası	Populasyon büyüklüğü	Nesiller	iterasyon	En iyi f(x) değeri	X_1 ve X_2	Çözüm Süresi(sn)
1. çalıştırma	50	1	1100	7,699	X_1=22,84 X_2=6,995	1,0352
		2	2150	8,847		
		3	3200	9,048		
		4	4250	9,762		
		5	5300	10,06		
2. çalıştırma	50	1	1100	7,808	X_1=26,15 X_2=6,998	1,1136
		2	2150	8,994		
		3	3200	9,5		
		4	4250	9,835		
		5	5300	10,41		
3. çalıştırma	50	1	1100	8,043	X_1=22,96 X_2=6,999	1,5741
		2	2150	8,536		
		3	3200	9,357		
		4	4250	9,717		
		5	5300	10,08		
4. çalıştırma	50	1	1100	7,946	X_1=29,72 X_2=6,96	1,0991
		2	2150	8,703		
		3	3200	9,433		
		4	4250	9,956		
		5	5300	10,72		
5. çalıştırma	50	1	1100	7,749	X_1=22,52 X_2=6,996	1,1090
		2	2150	8,601		
		3	3200	9,103		
		4	4250	9,522		
		5	5300	10,03		

Çizelge 7.6'da detayları verilen çalışmada, populasyon büyüklüğü 50 olarak belirlenip, genetik algoritma çözümü matlab ga fonksiyonu ile sağlanmış ve aşağıdaki sonuçlar elde edilmiştir.

Adım(step) sayısı ortalama $X_1 = 25$ olarak bulunmuş,
Kullanım kaynaklı hata ise ortalama $X_2 = 7$ olarak bulunmuştur.

Bu verilere göre yeni bir sürüm yayınlamaya gerek kalmadan programı kullanacak en fazla kullanıcı sayısı 10 olarak bulunmuştur. Ortalama çözüm süresi ise 1,1862 sn. tespit edilmiştir.

Çizelge 7.7 Nesillere ait hesaplamalar (Populasyon 100)

Çalıştırma sırası	Populasyon büyüklüğü	Nesiller	iterasyon	En iyi f(x) değeri	X_1 ve X_2	Çözüm Süresi(sn)
1. çalıştırma	100	1	2200	7,838	X_1=38,15 X_2=6,98	1,8799
		2	4300	9,011		
		3	6400	9,855		
		4	8500	10,84		
		5	10600	11,46		
2. çalıştırma	100	1	2200	8,103	X_1=40,75 X_2=6,959	1,6869
		2	4300	9,083		
		3	6400	10,14		
		4	8500	11,04		
		5	10600	11,65		
3. çalıştırma	100	1	2200	8,28	X_1=42,06 X_2=7,00	1,7144
		2	4300	9,334		
		3	6400	10,29		
		4	8500	11,11		
		5	10600	11,78		
4. çalıştırma	100	1	2200	8,158	X_1=39,36 X_2=6,996	1,7237
		2	4300	9,411		
		3	6400	10,28		
		4	8500	10,91		
		5	10600	11,56		
5. çalıştırma	100	1	2200	8,542	X_1=40,22 X_2=6,989	1,7245
		2	4300	9,713		
		3	6400	10,21		
		4	8500	10,85		
		5	10600	11,63		

Çizelge 7.7'de detayları verilen çalışmada, populasyon büyüklüğü 100 olarak belirlenip, genetik algoritma çözümü matlab ga fonksiyonu ile sağlanmış ve aşağıdaki sonuçlar elde edilmiştir.

Adım(step) sayısı ortalama $X_1 = 40$ olarak bulunmuş,
Kullanım kaynaklı hata ise ortalama $X_2 = 7$ olarak bulunmuştur.

Bu verilere göre yeni bir sürüm yayınlamaya gerek kalmadan programı kullanacak en fazla kullanıcı sayısı 12 olarak bulunmuştur. Ortalama çözüm süresi ise 1,7459 sn. tespit edilmiştir.

Çizelge 7.8 Nesillere ait hesaplamalar (Populasyon 150)

Çalıştırma sırası	Populasyon büyüklüğü	Nesiller	iterasyon	En iyi f(x) değeri	X_1 ve X_2	Çözüm Süresi(sn)
1. çalıştırma	150	1	3300	8,295	X_1=42,42 X_2=6,974	2,2773
		2	6450	9,818		
		3	9600	10,62		
		4	12750	11,52		
		5	15900	11,79		
2. çalıştırma	150	1	3300	8,803	X_1=42,37 X_2=6,982	2,4517
		2	6450	9,962		
		3	9600	10,95		
		4	12750	11,73		
		5	15900	11,79		
3. çalıştırma	150	1	3300	8,318	X_1=42,33 X_2=6,987	2,2709
		2	6450	9,432		
		3	9600	10,47		
		4	12750	11,31		
		5	15900	11,79		
4. çalıştırma	150	1	3300	8,326	X_1=42,31 X_2=6,991	2,2814
		2	6450	9,573		
		3	9600	10,58		
		4	12750	11,41		
		5	15900	11,79		
5. çalıştırma	150	1	3300	8,359	X_1=42,28 X_2=6,996	2,3472
		2	6450	9,849		
		3	9600	10,72		
		4	12750	11,47		
		5	15900	11,79		

Çizelge 7.8'de detayları verilen çalışmada, populasyon büyüklüğü 150 olarak belirlenip, genetik algoritma çözümü matlab ga fonksiyonu ile sağlanmış ve aşağıdaki sonuçlar elde edilmiştir.

Adım(step) sayısı ortalama $X_1 = 42$ olarak bulunmuş,
Kullanım kaynaklı hata ise ortalama $X_2 = 7$ olarak bulunmuştur.

Bu verilere göre yeni bir sürüm yayınlamaya gerek kalmadan programı kullanacak en fazla kullanıcı sayısı 12 olarak bulunmuştur. Ortalama çözüm süresi ise 2,3257 sn. tespit edilmiştir.

7.5. Doğrusal olmayan problem için bulgular

Doğrusal olmayan fonksiyon ve kısıt içerecek şekilde kurgulanan problem fmincon fonksiyonu ve genetik algoritma ile ayrı ayrı çözülmüştür. Genetik algortima yaklaşımı sezgisel olduğundan çözüm için yazılan program parametreleri değiştirilerek beşer kez çalıştırılmıştır. Elde edilen sonuçlar çizelge 7.6, 7.7 ve 7.8'de gösterimiştir. Genetik algoritma parametre değişikliği populasyon büyüklüğünde yapılmış ve sırasıyla 50, 100 ve 150 olarak tanımlanan populasyonlar çözüm için kullanılmıştır. Fmincon ve genetik algoritma sonuçları çizelge 7.9'da verilmiştir.

Çizelge 7.9 Fmincon ve Genetik Algoritma çözüm sonuçları

	X_1	X_2	F(X)	Çalışma süresi (sn)
Fmincon	64	4	12	0,0286
GA 50	25	7	10	1,1862
GA 100	40	7	12	1,7459
GA 150	42	7	12	2,3257

Çizelge 7.9'da görüldüğü gibi, genetik algoritma sonuçları populasyon büyüklüğüne bağlı olarak değişmektedir. Populasyon büyüklüğü arttıkça sonuçların çözüme yaklaştığı görülmekte, bununla ilişkili olarak çalışma sürelerinde de artış gözlemlenmektedir.

8. SONUÇLAR

Genetik Algoritma ile optimizasyon, canlılardaki biyolojik işleyişin örneklemesi ile elde edilmektedir. Belirlenen büyüklükte bir topluluk tasarlayıp ve bu topluluğun geçirdiği evrim sürecini incelemek, Genetik Algoritma işleyişini somutlaştırmak açısından iyi bir yaklaşım olacaktır.

Yazılım ve yazılım proje yönetimi, özellikle son yıllarda önem kazanan bir konudur. İş gücü ve bütçe planlaması için proje planlamasının doğru yapılması gerekmektedir. Bütçesinde, kalitesinde ve zamanında gerçekleşen bir proje, ancak ve ancak iyi bir proje yönetimi ile mümkündür. Analiz, tahmin ve planlama gibi analitik süreçler içeren proje yönetimi konusu, matematiksel olarak tanımlanabilir ve çözümü gerçekleştirilebilir.

Bu çalışmada, doğrusal ve doğrusal olmayan optimizasyon problemlerinin çözümünde Genetik Algoritma tekniğinin kullanımı incelenmiştir. Optimizasyon problemi olarak yazılım proje yönetimi konusu incelenmiş ve bu konu ile ilgili problemler ele alınmıştır. Bu çalışma neticesinde yazılım projelerinin bir optimizasyon problemi olarak tasarlanabileceği ve çözümü için Genetik Algoritma yaklaşımının kullanılabileceği görülmüştür.

Elde edilen sonuçlar değerlendirildiğinde, genetik algoritma çözümünün, sezgisel olmayan klasik çözümlere yakın değerler elde ettiği görülmüştür. Gerçekleştirilen denemelerde, populasyon büyüklüğü arttıkça klasik çözümlerin ürettiği değerlere daha çok yaklaşıldığı, ancak çalışma süresinin arttığı tespit edilmiştir.

KAYNAKLAR

Albayrak, B. (2005). *Proje Yönetimi.* Ankara: Nobel.

Altıparmak, F., Dengiz, B., Smith, A. (1998). Reliability Optimization of Computer Communication Networks Using Genetic Algorithms. *IEEE International Conference on Systems, Man, and Cybernetics*, (s. Cilt 5, 4676-4681).

Bagad, V. (2008). *Management&Finance.* Tehcnical Publication.

Biegal, J., Davern, J. (1990). *Genetic Algorithms And Job Scheduling.* Orlando, USA: University of Central Florida,Computers and Industrial Engineering.

Bittner, K., Spence, I. (2006). *Managing Iterative Software Development Projects.* Addison Wesley.

Boehm, B. (1989). *Software Risk Management.* IEEE Computer Society.

Boehm, B. W. (1988). *A Spiral Model of Software Development and Enhancement.* IEEE Computer.

Brooks, F. (1995). *The MythicalManMonth, Essays on Software Engineering.* Addison Wesley.

Davis, L. (1985). Job Shop Scheduling With Genetic Algorithm. *Proceeding of the first International Conference on Genetic Algorithms* (s. 136-140). Pittsburgh, Pennsylvania: Carnegie Mellon University.

Davis, L. (1991). *Handbook of Genetic Algorithms.* New York: Van Nostrand Reinhold.

Deming, W. (1986). *Out of the Crisis.* Cambridge: MIT Center for Advanced Engineering.

Dipankar, D., Zbigniew, M. (1997). *Evolutionary Algorithms In Engineering Applications.* Germany: Springer.

Drucker, P. (2006). *Klasik Drucker.* Bahçeşehir Üniversitesi Yayınları.

Elmas, P. (2011). *Yapay Zeka Uygulamaları.* Ankara: Seçkin.

Fairley, R. (2009). *Managing and Leading Software Project.* Wiley.

Fayol, H. (2012). *Genel ve Endüstriyel Yönetim* (3. Baskı 2012 b.). (M. Çalıkoğlu, Çev.) Adres Yayınları.

Goldberg, D. (1989). *Genetic Algorithms In Search, Optimization And Machine Learning.* USA: Addison-Wesley Publishing Company, Inc.

Gonzales, E., Fernandez, M. (2000). *Genetic Optimisation Of A Fuzzy Distribution Model.* International Journal Of Physical Distribution & Logistics Management.

Güney, D. D. (2007). *Yönetim ve Organizasyon.* Nobel Yayın Dağıtım.

Holland, J. (1975). *Adaptation in Natural and Artificial System.* Ann Arbor, MI: Univercity of Michigan Press.

McConnell, S. (2004). *Professional Software Development.* Boston: Addison-Wesley.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structure = Evolution Programs.* Berlin, Germany: Springer-Verlag.

Nizam, D. A. (2014). *Yazılım Proje Yönetimi.* İstanbul: Papatya Yayıncılık Eğitim.

Ören, T., Üney, T., Çölkesen, R. (2006). *Türkiye Bilişim Ansiklopedisi.* Papatya.

Pressman, R. (2009). *Software Engineering: A Practitioner's Approach.* McGraw-Hill Science.

Shewhart, W. (1931). *Economic Control of Quality of Manufactured Product.* New York: Van Nostrand.

Taşkın, D. Ç., Emel, Y. G. (2009). *Sayısal Yöntemlerde Genetik Algoritmalar.* Alfa Aktüel.

Tiwari, M., Vidyarthi, N. (2000). Solving Machine Loading Problems In A Flexible Manufacturing System Using A Genetic Algorithm Based Heuristic Approach. *International Journal Of Production Research, Volume: 38, Number: 14*, 3369.

Vishnu, P. (2007). *Principles of Management.* United States: Computech Publication Limited.

Winston, R. (1970). *Managing the Development of Large Software Systems.* Proceedings of IEEE Wescon.

Wong, M., Leung, K. (1999). *Data Mining Using Grammar Based Genetic Programming and Applications.* London: Kluwer Academic Publishers.

Yeniay, Ö. (2001). An Overview Of Genetic Algorithms. *Anadolu Üniversitesi Bilim ve Teknoloji Dergisi*, Cilt 02, Sayı 1.

EKLER

EK A. Doğrusal programlama için yazılan matlab kodları

EK B. Doğrusal problem için simple_fitness amaç ve simple_constraint kısıt fonksiyonları

EK C. Doğrusal problemin Genetik algoritma çözümü için yazılan matlab kodları

EK D. Doğrusal olmayan problem için Nl_func amaç ve nl_cons kısıt fonksiyonları

EK E. Doğrusal olmayan problemin fmincon fonksiyonu ile çözümü için yazılan matlab kodları

EK F. Doğrusal olmayan problemin genetik algoritma çözümü için yazılan matlab kodları

EK A. Doğrusal programlama için yazılan matlab kodları

```
tic;
% A matrisi kısıt fonksiyonlarındaki büyüklük küçüklük
eşitsizliklerine
% göre doldurulur. -1 ve +1 kısıt fonksiyonuna göre verrilir
% lineer eşitsizlik kısıtları için matris
A = [-1, -1, -1, 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, -1, -1, -1, 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, -1, -1, -1, 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, -1, -1, -1, 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, -1, -1, -1, 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, -1, -1, -1, 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, -1, -1,
...
      -1, 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,
...
      -1, -1, -1, 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, -1, -1, -1, 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, -1, -1, -1;
      1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 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, 1, 0, 0, 1, 0, 0, 1, 0, 0,
...
      1, 0, 0, 1, 0, 0, 1, 0, 0;
      0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0,
...
      0, 1, 0, 0, 1, 0, 0, 1, 0;
      0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,
...
      0, 0, 1, 0, 0, 1, 0, 0, 1;
      -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 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, -1, 0, 0, -1, 0, 0, -1, 0,
...
      0, -1, 0, 0, -1, 0, 0, -1, 0, 0;
      0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0,
...
      -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0;
```

```
      0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0,
...
      0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1]

%min amaç fonksiyonunun değerleri yazılır.
%doğrusal amaç fonksiyonu vektörü
f = [3192; 2964; 2508; 2296; 2240; 1792; 2448; 1872; 1632; 2756;
1976; ...
     1768; 1960; 1848; 1512; 2256; 1248; 1344; 2964; 2204; 2356;
3096; ...
     2520; 2160; 2240; 1280; 1040; 2244; 1584; 1100]

%kısıt fonksiyonlarının sonuçları yazılır
% lineer eşitsizlik kısıtları içi vektör
b = [-8; -8; -8; -8; -16; -16; -16; -16; -16; -16; 86; 100; 80; 64;
...
     -82; -80; -56; -60]

%değişken sayısı yazılır
%alt sınır vektor
lb = zeros(30,1)

[x,fval,exitflag,output,lambda] = linprog(f,A,b,[],[],lb);

round(fval)

x

toc;
```

EK B. Doğrusal problem için simple_fitness amaç ve simple_constraint kısıt fonksiyonları

```
Fonksiyon simple_fitness:
function y = simple_fitness(x)
    y = 3192*x(1) + 2964*x(2) + 2508*x(3) + 2296*x(4) + 2240*x(5)...
    + 1792*x(6) + 2448*x(7) + 1872*x(8) + 1632*x(9) + 2756*x(10)...
    + 1976*x(11) + 1768*x(12) + 1960*x(13) + 1848*x(14) +
1512*x(15)...
    + 2256*x(16) + 1248*x(17) + 1344*x(18) + 2964*x(19) +
2204*x(20)...
    + 2356*x(21) + 3096*x(22) + 2520*x(23) + 2160*x(24) +
2240*x(25)...
    + 1280*x(26) + 1040*x(27) + 2244*x(28) + 1584*x(29) +
1100*x(30);

Fonksiyon simple_constraint:
function [c, ceq] = simple_constraint(x)

c(1) = x(1) + x(2) + x(3) - 8 ...
    + x(4) + x(5) + x(6) - 8 ...
    + x(7) + x(8) + x(9) - 8 ...
    + x(10) + x(11) + x(12) - 8 ...
    + x(13) + x(14) + x(15) - 16 ...
    + x(16) + x(17) + x(18) - 16 ...
    + x(19) + x(20) + x(21) - 16 ...
    + x(22) + x(23) + x(24) - 16 ...
    + x(25) + x(26) + x(27) - 16 ...
    + x(28) + x(29) + x(30) - 16;

c(2) = x(1) + x(4) + x(7) + x(10) - 86 ...
        + x(13) + x(16) + x(19) + x(22) + x(25) + x(28) - 100 ...
        + x(2) + x(5) + x(8) + x(11) + x(14) + x(17) + x(20) + x(23)
...
        + x(26) + x(29) - 80 ...
        + x(3) + x(6) + x(9) + x(12) + x(15) + x(18) + x(21) + x(24)
...
        + x(27) + x(30) - 64;

c(3) = x(1) + x(4) + x(7) + x(10) - 82 ...
       + x(13) + x(16) + x(19) + x(22) + x(25) + x(28) - 80 ...
       + x(2) + x(5) + x(8) + x(11) + x(14) + x(17) + x(20) ...
       + x(23) + x(26) + x(29) - 56 ...
       + x(3) + x(6) + x(9) + x(12) + x(15) + x(18) + x(21) + x(24)
...
       + x(27) + x(30) - 60;
```

EK C. Doğrusal problemin Genetik algoritma çözümü için yazılan matlab kodları

```
ceq = [];

%üreme fonksiyonu olarak lineer uygulanabilir fonksiyonu seçilmiş
options = gaoptimset('CreationFcn',@gacreationlinearfeasible)

%genetik algoritma özelliklerinin seçilmesi
%çaprazlama için "iki nokta" çaprazlama seçilmiştir.
options = gaoptimset(options,'CrossoverFcn',@crossovertwopoint)


%seçim işlemi olarak turnuva yöntemi seçilir.
%Turnuva boyutu en az 2 olmalıdır. Turnuva boyutunun varsayılan
değeri
%4'tür.
options = gaoptimset(options,'SelectionFcn',...
                     {@selectiontournament,4})

%hibrid kısmında pattern search seçilmiştir
options = gaoptimset(options,'HybridFcn',...
                      @patternsearch)


tic;

A = [-1, -1, -1, 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, -1, -1, -1, 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, -1, -1, -1, 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, -1, -1, -1, 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, -1, -1, -1, 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, -1, -1, -1, 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, -1, -1,
...
      -1, 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,
...
      -1, -1, -1, 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, -1, -1, -1, 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, -1, -1, -1;
```

```
      1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 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, 1, 0, 0, 1, 0, 0, 1, 0, 0,
...
      1, 0, 0, 1, 0, 0, 1, 0, 0;
      0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0,
...
      0, 1, 0, 0, 1, 0, 0, 1, 0;
      0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,
...
      0, 0, 1, 0, 0, 1, 0, 0, 1;
      -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 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, -1, 0, 0, -1, 0, 0, -1, 0,
...
      0, -1, 0, 0, -1, 0, 0, -1, 0, 0;
      0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0,
...
      -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0;
      0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0,
...
      0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1]

b = [-8; -8; -8; -8; -16; -16; -16; -16; -16; -16; 86; 100; 80; 64;
...
     -82; -80; -56; -60]

lb = zeros(30,1);

[x,fval,exitflag,output] = ga(@simple_fitness,...
    30,A,b,[],[],lb,[],[],options)


round(fval)

fprintf('The number of generations was : %d\n', output.generations);
fprintf('The number of function evaluations was : %d\n',
output.funccount);
fprintf('The best function value found was : %g\n', fval);

x

toc;
```

EK D. Doğrusal olmayan problem için Nl_func amaç ve nl_cons kısıt fonksiyonları

```
function f=nl_func(x)

f = -x(1)^0.5 - (2*x(2)^0.5);


function [c,ceq]=nl_cons(x)

% Nonlinear inequality constraints
c1 = x(2) - 7;
c2 = 2*x(1)^0.5 + x(2) - 20;

c=[c1;c2];

% Nonlinear equality constraints
ceq = [];
```

EK E. Doğrusal olmayan problemin fmincon fonksiyonu ile çözümü için yazılan matlab kodları

```
tic;
A   =  [];
B   =  []; % the linear inequality constraints: A*X <= B
Aeq =  [];
Beq =  []; % the linear equality constraints: Aeq*X = B
LB  =  [];
UB  =  []; % LB <= X <= UB
X0  =  [1, 1]; % initial guess

[X,FVAL,EXITFLAG,OUTPUT,LAMBDA] =
fmincon(@nl_func,X0,A,B,Aeq,Beq,...
LB,UB,@nl_cons);

toc;
```

EK F. Doğrusal olmayan problemin genetik algoritma çözümü için yazılan matlab kodları

```
tic;
ObjectiveFunction = @nl_func;
nvars = 2;    % Number of variables

lb = [0 0]; % Lower bound
ub = [inf inf]; % Upper bound

ConstraintFunction = @nl_cons;


options = gaoptimset('PopulationSize',150,...
    'Generations', 20, ...
    'EliteCount', 4, ...
    'MutationFcn',@mutationadaptfeasible, ...
    'TolFun', 1e-6, ...
    'PlotFcns', @gaplotbestf, ...
    'Display','iter');



[x, fval, exitflag, display, output] = ga(ObjectiveFunction, ...
                    nvars, [], [], [], [], ...
                    lb, ub, ConstraintFunction,options);


round(x);
round(fval);

toc;
```

ÖZGEÇMİŞ

Adı Soyadı : Yücel Dil

Doğum Yeri ve Yılı : Erzincan, 17/11/1981

Medeni Hali : Evli

Yabancı Dili : İngilizce

E-posta : yuceldil@gmail.com

Eğitim Durumu

Lisans : Trakya Üniversitesi,
Mühendislik Mimarlık Fakültesi, Bilgisayar Mühendisliği Bölümü

Yüksek Lisans : İstanbul Ticaret Üniversitesi,
Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı

Mesleki Deneyim

Mega Yazılım, Yazılım Uzmanı	2004-2006
SFS, Yazılım Uzmanı	2006-2007
Hayat Holding, Yazılım Uzmanı	2007-2011
Doğan Holding, Kıdemli Yazılım Uzmanı	2011-2014
Anel Holding, Uygulama Geliştirme Yöneticisi	2014-(Devam ediyor)