OPTİMİZASYON YÖNTEMİ OLARAK GENETİK ALGORİTMALAR

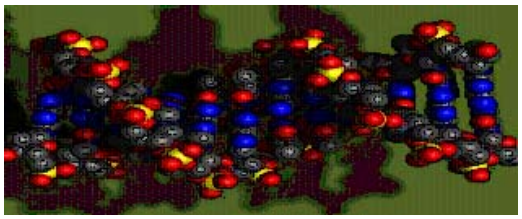
Dr. M. Turhan ÇOBAN
EGE Üniversitesi, Mühendislik Fakultesi, Makine Mühendisliği bölümü,
Bornova, İZMİR

1. ÖZET

Java Bilgisayar programlama ile sayısal çözümleme kitabında, genetik algoritmalar da dahil olmak üzere çeşitli sayısal yöntemlere bilgisayar programlama örnekleri ile yer verdim, bu kitaptan göreceli yeni olan bir kavramı genetik algoritmalar kullanarak optimizasyon yöntemlerine değinmeye çalışacağım. Genetik algoritmalar evrim programlaması denilen ve günümüzde suni zeka alanında oldukça yoğun kullanılan bir programlama tekniğidir. Adından da anlaşılacağı gibi temelini evrim kuramından almıştır. Temel olarak evrim prosesi en iyi uyumu sağlayan bireylerin hayatta kalması prensibine dayanır. Biz bu prensibi optimizasyon prosesine maksimum değer sağlayan bireylerin hayatta kalması olarak adapte edeceğiz. Temel prensiplerive hazırlanan bilgisayar kodlarını sunacağız

2. GİRİŞ

Genetik algoritmalar canlı yapıların Darwin teorisinde belirtildiği gibi doğaya uyum hayatta kalma) mekanizması içinde kendi kendilerini genlerdeki değişmeyle değiştirmelerinin fonksiyonlara optimizasyon problemi olarak uyarlanmış halidir. Darwin teorisi anne ve babaların özelliklerinin çocuklara geçtiğini, bazen de anne ve babada olmayan yeni bazı özelliklerin mutasyon denilen mekanizmayla oluştuğunu, bu özellikler türün yaşama şansını arttırıyorsa bu bireylerin çoğalarak bu özellikleri kendi çocuklarına geçirdiği, özellikler başarısızsa ölecekleri için başarısız özellikleri gelecek nesillere geçiremeyecekleri böylece ancak doğaya uyumda başarılı olan özelliklerin daha sonraki nesillerde birikmesi sonucu doğaya uyum sağlayan türlerin oluşabileceğini getirmiştir. Daha sonra Genetik biliminin gelişmesiyle bu prensibin altındaki biokimyasal yapılar (genler) bulunmuş ve bu sistemin matematiksel ve istatistiksel olarak nasıl çalıştığı daha detaylı anlaşılmıştır. Genler 4 amino asitin yapısı içinde sıralanması ile canlı yapıların özelliklerinin anahtarını oluştururlar. Bu özellikler, anne ve babadan gelen genlerle çocuklara aktarıldığı gibi bu aktarma prosesinde zaman zaman oluşabilen hatalarda Darwin'in tanımladığı mutasyon prosesini oluşturmaktadır.



Şekil 2.1 DNA'nın yapısı

Tüm canlı yapılar hücreler içerir. Her hücre bir veya daha fazla kromozom seti barındırır. Kromozomlar DNA dizinleridir. Ve tüm organizmanın modelini içlerinde barındırırlar. Kromozom DNA'lardan oluşan gen bloklarını barındırır. Her genin belli bir özelliğin kodunu taşıdığı söylenebilir, örneğin göz rengi gibi. Bu özelliklerin alabildiği değerlere de gen değerleri adı verilir(örneğin kahverengi göz, yeşil göz). Kromozomların hepsi bir arada canlının karekterlerini oluşturur, buna cenom denir. Cenomdaki belli bir özelliği oluşturan guruba cenotip ismi verilir. Canlıların doğumdan sonra çevresel etkiler yoluyle oluşan özeliklerine de fenotip ismi verilir.

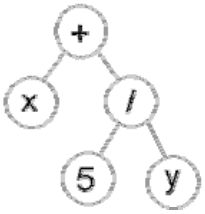
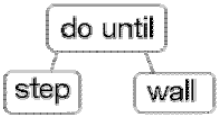
Üreme esnasında önce rekombinasyon oluşur, anne ve babaları genleri bölünerek birleşirler ve yeni bir gen yapısı oluştururlar. Yeni oluşan genlerin bir kısmı mutasyona da uğrayabilir, mutasyon gen DNA dizilimindeki anne ve babadan genlerin kopyelenmesi sırasında oluşan hatalardır, bunlar yeni bir dizilim oluşmasına sebep olurlar. Doğal Uyum organizmanın hayatta kalma başarısı ile ölçülen ve gen guruplarının direk olarak etkilediği bir olaydır. Mutasyon çoğunlukla genin fonksiyonlarını bozduğundan bireylerin hayatta kalma olasılıklarını düşürür ve ölmelerine yol açar. Fakat bazı genlerde mutasyon sonucu oluşan özellikler türün hayatta kalma olasılığını yükseltebilir, bu bireyler hayatta kalma olasılıklarının yükselmesinden dolayı bu özellikleri bir sonraki nesle iletme olasılıkları da yükselmiş olacaktır.

Evrimsel hesaplama ilk defa 1960'lı yıllarda I Rechenberg'in "*Evolutionsstrategie* in original " evrim statejileri çalışmasında yer almıştır. Fikir daha sonra başka araştırıcılarca geliştirilmiştir. Genetik algoritmalar John Holland ve öğrencileri tarafından ilk defa bu günkü anlamıyla oluşturulmuştur. 1975 yılında basılan "*Adaption in Natural and Artificial Systems*" kitabında yazar bu tekniğin nasıl kullanılacağını açıklamıştır

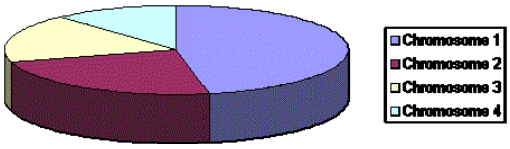
Genetik algoritmalar yukarda değindiğimiz mekanizmanın yeni nesiller üretmekte ve yeni nesillerin doğaya uyum mekanizmasını açıklayan Darwin teorisinden yola çıkarak oluşturulmuştur. Algoritma kromozomlardan oluşan bir gurup çözüm setinden oluşur. Bir gurubun oluşturduğu çözüm setlerinden problemin karekteristiğine göre seçilen çözüm setleri üreme yolu ile yeni nesillere taşınır. Başarılı bireylerin daha fazla üreme özellikler oluşabilir, bu özellikleri bir kısmı bu bireylerin başarısını azaltırken bazen de daha başarılı yeni bireyler de oluşabilir. Genetik algoritmaların temel yapıları şu şekilde özetlenebilir :

[BAŞLA] Probleme göre n kromozondan oluşan bir set oluştur
[UYUM] x genlerinden oluşan her kromozonun *f(x)* uyum değerini *tüm nüfus için hesapla*
[YENİ NESİL] Alttaki stepler uyarınca yeni bir nesil oluştur
[SEÇİM] Toplam nüfustan uyumları göz önünde bulundurularak bir anne baba çifti seçin, bu seçimde uyum daha fazla ise seçim olasılıkları da buna göre fazlalaşsın.
[ÜREME] Üreme yoluyla yeni bir nesil üretip anne babalarıyla bu yeni nesli değiştirn. Eğer üreme olmadı ise eski nesli koruyun.
[MUTASYON] belli bir olasılık içinde mutasyonları oluşturun
[KABUL] Yeni bireyleri nüfusa ekleyin
[DEĞİŞTİRME] yeni nesli aynı şekilde tekrar yeni nesiller elde etmek için kullanın
[TEST] eğer istenilen şartlar oluşmuşsa simulasyonu durdurun
[DÖNGÜ] Oluşmamışsa geri dönerek işleme devam edin.

Gen veya kromozon genetik hesaplamada kullanılan temel elemanlardır. Matematiksel olarak düşünürsek n değişkenli bir sistemde her değişkeni bir gen olarak düşünürüz. Tüm değişken setini bütün olarak kromozon olarak adlandırabiliriz. Kromozonların yapıları

Kromozom A **kromozon B** **kromozon C**	**5.3243 0.4556 2.3293 2.4545** **ABDJEIFJDHDIERJFDLDFLFEGT** **(geri), (geri), (sağ), (ileri), (sol)**
Kromozom D	 (+ x (/ 5 y))
Kromozom E	 (do_until step wall)

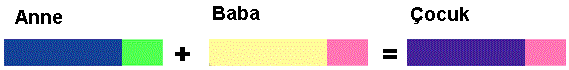
Gibi birbirinden çok farklı eleman tipleri kullanılarak oluşturulabilir. Doğadaki gerçek genlerde uyum dediğimizde doğada hayatta kalabilme yeteneğini anlıyoruz. Matematiksel olarak optimizasyon(maksimizasyon) için bu kavramı kullanacak olursak burada fonksiyonun değeridir. Daha sonra bu uyum değerleri toplanarak yüzde şekline çevrilebilir.



Bu yüzdeye çevrilmiş uyum değer fonksiyonlarının arasından tesadüfi olarak seçim yaparsak,
tesadüfi seçimde sayıların toplanmış yüzde olasılığı daha iyi uyum gösteren (maksimum fonksiyon değerini alan) birey için daha yüksektir. Uyum gösteren bireyler arasından tesadüfi olarak seçilen kısmının bir sonraki nesli üretmesine(çoğalmasına) izin verilir. Çoğalma prosesi anne ve baba diyebileceğimiz iki bireyin genlerinin birleşerek yeni bir birey (çocuk) oluşturmasıdır. Matematiksel olarak buna çaprazlama prosesi denir. Çaprazlama anne ve baba genlerinin bir kısmının (tesadüfi olarak) seçilerek bir araya getirilmesi ile gerçekleşir.
Örneğin binari rakamlardan oluşan bir gen için : Eğer tek çaprazlama metod kullanırsak, bu metodda tek bir tesadüfi sayı gen gurubunu belli bir oranda böler. Aynı oranda bölünen genin bir kısmı anneden diğer kısmı da babadan alınır. Ve yeni bireyi oluşturur. Genlerin bölme oranı tesadüfi olarak belirlenir.

Örneğin 8 bitten oluşan bir gurup için:
8*random(0-1) = 5 ise ilk 5 bitti anneden, son 3 biti babadan alarak yeni bireyi oluştururuz.

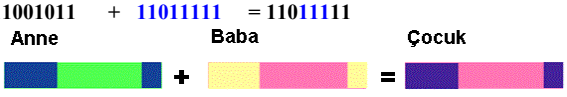
11001011 + 11011111 = 11001111



Kromozom 1 (Anne)	11011 | 00100110110
Kromozom 2 (Baba)	11011 | 11000011110
1. Çocuk	11011 | 11000011110
2. Çocuk	11011 | 00100110110

Eğer çift çaprazlama uygulamak istersek, bu metodda iki tesadüfi sayı gen gurubunu belli bir oranda üç parçaya böler. Aynı oranda bölünen genin bir kısmı anneden diğer kısmı da babadan sonra kısım tekrar anneden alınır (veya tersi yapılır). Ve yeni bireyi oluşturur. Genlerin bölme oranı tesadüfi olarak belirlenir.
Örneğin 8 bitten oluşan bir gurup için:
Random1=8*random(0-1) = 2
Random2=8*random(0-1) = 6
ise ilk 2 biti anneden, 3 den 6ıncı bite kadar babadan ve 7 ve 8 inci biti tekrar biti babadan alarak yeni bireyi oluştururuz.

11001011 + 11011111 = 11011111



Eğer düzenli çaprazlama uyulamak istersek, bu metodda her gen tesadüfi olarak anneden veya babadan seçilerek yeni birey(çocuk) oluşturulur.
Örneğin 8 bitten oluşan bir gurup için:
Random1=random(0-1) 0.5 den küçük ise 1 inci gen olarak annenin geni, 0.5 den büyük ise babanın geni seçilir.
11001011 + 11011111 = 11001011

3.MATAMATİKSEL OPTİMİZASYON

Biz burada sizlere bu metodun bir matematik aracı olarak optimizasyon (minimizasyon – maksimizasyon) problemlerine uygulanmasını ve yöntemin temel çalışma prensibini açıklamaya çalışacağız.

Optimizasyon (Minimizasyon-Maksimizasyon) problemi Matematiksel olarak
f(x1,x2,x3,…..xi) fonksiyonunun maksimumunu bulma olarak tanımlanabilir.
Bu işlemi oluştururken önce maksimum ve minimum işleminin aynı işlem olduğunu unutmıyalım

max [f(x)] = min [- f(x)] = min [g(x)]

Fonksiyon oluşturmada göz önüne alınması gereken diğer bir konu da fonksiyon değerine sabit bir değer ilave ettiğimizde maximum değerin yerinin değişmiyeceğidir.

max [f(x)] = max [f(x) + C] = max [g(x)]

Bu özelliği kullanarak fonksiyonumuzun çözümünün her zaman pozitif bölgeye düşmesini sağlıyabiliriz.

Mühendislik optimizasyonunda çözdüğümüz sayıların çoğu gerçek sayılar olmak zorundadır. Ancak bu gerçek bir sayıyı bir binari sayılar dizini olarak ifade etmek her zaman mümkündür. Problemi optimizasyon problemi olarak düşündüğümüzde bir gurup bağımsız değişkenimiz mevcuttur
(x0,x1,x2,..xi..xn)
Genetik algoritmaları uygulamak için her bağımsız değişkenin sınır değerlerinin (maksimum ve minumum) bilinmesi zorunludur. Bu yüzden değişken gurubumuz için
xo : xo_minimum xo_maximum
x1 : x1_minimum x1_maximum
….
xi : xi_minimum xi_maximum
…
xn : xn_minimum xn_maximum
Tanımları yapılır.
Her bağımsız değişken için ayrıca değişkenin binari eşdeğerinin kaç bir olacağının tanımlanması gerekir. Bağımsız değişkenimiz
xi_min <= xi <= xi_max N bit binary
Olarak tanımlanmış ise :
Binary eşdeğeri veya binari olarak verilmiş sayının gerçek sayı eşdeğeri iki adımlık bir prosesle hesaplanabilir.
N digitlik bir binary gurubu tamsayıya

$$(b_n, b_{N-1}, b_{N-2}, \ldots b_1, b_0)_2 = (\sum_{i=0}^{N-1} b_i 2^i)_{10} = x'$$

De
Buna karşı gelen gerçek sayı ise

$$x = x\_min + x'\left(\frac{x\_max - x\_min}{2^N - 1}\right)$$

İfadesiyle hesaplanır. Aynı işlem ters olarak yapılarak da gerçek sayıdan binari sayıya ulaşılabilir

Bağımsız değişkenler her biri bir gen olmak için NDEĞİŞKEN kapsayan bir gurupta genlerin tek bir uzun binari dizide toplanması ile veya geni ihtiva eden sınıfın dizini olarak ifade edilebilir. Dizinlerden elde edilen bağımsız değişkenler optimizasyonu yapılacak fonksiyonda girdi olarak kullanılarak fonksiyonun değeri hesaplanır. Programlama yapılırken değişik

fonksiyonlar girilebilecek şekilde yapılırsa kodlama açısından daha genel bir kod elde edebilirsiniz.

```
class fa extends f_xj
{
 double func (double x[])
 { return
(21.5+x[0]*Math.sin(4.0*Math.PI*x[0])+x[1]*Math.sin(20.0*M
ath.PI*x[1]));
 }
}
```

Bağımsız değişken ve fonksiyonun bu bağımsız değişken için aldığı değer bütün olarak bir cenotip oluşturur. Bir veya daha fazla cenotip kromozomlar olarak bir araya gelebilir. (bir veya birden fazla fonksiyon bir arada değerlendirilebilir). Kromozon sayılarını da birden fazla düşünebiliriz. Bu sistem hep birlikte toplumumuzun bireylerini oluşturur (Optimize edilecek denklem sistemi ve bağımsız değişkenler sınır değerleri)

Programlayıcı veya program kullanıcısı tarafından toplumuzun kaç bireyden oluşacağı önceden belirlenir. Bu sayıya eşdeğer birey bağımsız değişken değerleri tesadüfi sayılardan oluşan(sayını verilen limit değerleri içinde) bir set olarak oluşturulur. Fonksiyon değerleri de her cenotip için hesaplanır. (Mühendislik optimizasyon problemlerinde genelde çok değişkenli bir fonksiyon mevcuttur). Bu değerler genin uyum değerleridir. Tüm toplum bireylerinin uyum değerleri toplanarak toplam uyum hesaplanır. Sonra uyum değerleri toplam uyuma bölünerek göreceli uyum değerleri elde edilir. Göreceli uyum değerleri de birinci bireyden itibaren her birey için toplanarak toplanmış göreceli (kümülatif) uyum değerleri saptanır. Listedeki ilk bireyin toplanmış göreceli uyum değeri göreceli uyum değerine eşittir. Son bireyin ise bir'e eşittir.

Toplanmış göreceli uyum değerleri olasılıkları, uyumun büyüklüğüne göre oluşmuştur. Ve 0 ile 1 arasında yer alır. Eğer listemizden tesadüfi rakamlar aracılığı ile yeni bir nesil için seçme yapacak olursak tesadüfi rakamlar göreceli olarak eşit dağılacağından, daha yüksek uyuma sahip bireylerin seçilme şansı daha yüksektir. Aynı bireyin birden fazla kere seçilmesine de müsaade edilir.

Seçilen yeni toplum bireyleri arasında kullanıcılar tarafından belirlenen bir yüzde içerisinde kalan kısmının mutasyon prosesine girmesine izin verilir. Kullanıcı toplumdaki mutasyon yüzdesini tanımlar. Yine tesadüfi sayılar oluşturarak bu sayı mutasyon yüzdesinin altında ise o bireying genlerinin tesadüfi seçilen birisinin mutasyona uğraması (binari sistemde seçilen bitin değerinin değiştirilmesi sağlanır(1 ise 0, 0 ise 1 değerini alır)

Toplum yeniden değerlendirilmeye girer ve uyum, göreceli uyum değerleri saptanır, en iyi uyum sağlıyan birey seçilir. Bu bireyin değeri ayrı bir yerde tutularak tüm proses boyunca takip edilebilir.

2. BİLGİSAYAR KODUNUN OLUŞTURULMASI

Genetik algoritmalar doğadaki bu prosesi taklit ederler. Gen yapısını taklit etmek için bizim kurduğumuz modelde bilgisayar bitleri kullanılmıştır. Gene sınıfı gerekli prosesleri bit olarak yapar, eşdeğer olarak double değerini de hesaplayabiliriz, double-bit değişkenini kendi isteğimize göre yapabiliriz. Değişkenin bit uzunluğu da isteğe göre değişken olarak verilmiştir. Gene sınıfı değişkeni maksimum 64 bit (0-63) olmak üzere ayarlanabilir. Gene sınıfı içersinde çaprazlama ve mutasyon yapma metodlarını da barındırır. Mutasyon için seçilen 1 tesadüfi gen bitinin değeri değiştirilir(1 ise 0 0 ise 1 yapılır). Çaprazlama olarak yukarda belirtilen 3 çeşit çaprazlama mümkündür (düzenli çarprazlama, tek çaprazlama, çift çaprazlama). Sınıf double değerden veya "00110100111" gibi bir bit String değerinden de okuma yapabilir.

Program 3-1 Gene sınıfı

```
İmport java.util.*;
import java.io.*;

public class Gene extends Object
{
//Gene class generates digital(binary) array to represent a double
number
//actual binary defination changes depends on number of bits,
xmin and xmax
//binary data stored in b BitSet array
int N;        //number of binary digits(number of bits in a byte)
        //N should be less than 64
double xN;//Limit value for integer(2^N-1)
double xmin,xmax;//maximum and minimum limits of double
number
BitSet b; //binary array that strore the integer

public Gene(double ixmin,double ixmax,int iN)
{//randomly generate the gene
 N=iN;
 if(N>63) N=63;
 b=new BitSet(N);
 xN=Math.pow(2.0,N)-1;
 xmin=ixmin;
 xmax=ixmax;
 //generate random numbers to fill binary array
 setbits();
}

public Gene(String s,double ixmin,double ixmax)
{
//generate gene equal to a given string
 N=s.length();
 b=new BitSet(N);
 xN=Math.pow(2.0,N)-1;
 xmin=ixmin;
 xmax=ixmax;
 for(int i=N-1;i>=0;i--)
   if(s.charAt(i)=='1') b.set(N-1-i);
}

public Gene(double number,double ixmin,double ixmax,int iN)
{
//Generate a gene from a given double number
 N=iN;//number of binary digits(number of bits in a byte)
 if(N>63) N=63;
 b=new BitSet(N);
 xN=StrictMath.pow(2.0,N)-1;
 xmin=ixmin;
 xmax=ixmax;
 //z : integer equivalent of the double number
long z=(long)((number-xmin)/(xmax-xmin)*xN);
//System.out.println("inside double"+z);
long y=(long)xN;
double r=z;
for(int i=0;i<N;i++)
  {
  if((z-(z/2)*2)!=0) {b.set(i);}
  z/=2;
  }
}

public Gene()
{
//empty Gene
N=1;
b=new BitSet(N);
xN=StrictMath.pow(2.0,N)-1;
xmin=0;
xmax=1.0;
b.set(0);
}

public Gene(Gene g)
{
// copy a Gene from another one
copyGene(g);
```

```
}

public void copyGene(Gene g)
{
 N=g.N;
 b=(BitSet)g.b.clone();
 xN=Math.pow(2.0,N)-1;
 xmin=g.xmin;
 xmax=g.xmax;

}

public Gene copyGene()
{
 Gene g=new Gene(this);
 return g;
}

public Gene(Gene Father,Gene Mother)
{
//crosovertype will be selected randomly
double ix=Math.random();
if(ix<0.1) setGene(Father,Mother,"regular");
else if(ix>0.9) setGene(Father,Mother,"single");
else setGene(Father,Mother,"double");
}


public void  setGene(Gene Father,Gene Mother,String
crossovertype)
{
xmin=Father.xmin;
xmax=Father.xmax;
N=Father.N;
xN=Father.xN;
b=new BitSet(N);
int ir1,ir2,ir3;
if(crossovertype.equals("regular"))
{
for(int i=0;i<N;i++)
  {if(boolean_random()) { if(Father.b.get(i)) b.set(i);}
   else { if(Mother.b.get(i)) b.set(i);}
  }
}
else if(crossovertype.equals("single"))
{
   ir1=int_random();
   if(boolean_random())
   {   for(int i=0;i<ir1;i++)
         { if(Father.b.get(i)) b.set(i);}
       for(int i=ir1;i<N;i++)
         { if(Mother.b.get(i)) b.set(i);}
   }
   else
   {   for(int i=0;i<ir1;i++)
         { if(Mother.b.get(i)) b.set(i);}
       for(int i=ir1;i<N;i++)
         { if(Father.b.get(i)) b.set(i);}
   }
}
else if(crossovertype.equals("double"))
{
   ir1=int_random();
   ir2=int_random();
   int ix;
   if(ir1>ir2) {ix=ir2;ir2=ir1;ir1=ix;}
   if(boolean_random())
   {   for(int i=0;i<ir1;i++)
         { if(Father.b.get(i)) b.set(i);}
       for(int i=ir1;i<ir2;i++)
         { if(Mother.b.get(i)) b.set(i);}
       for(int i=ir2;i<N;i++)
         { if(Father.b.get(i)) b.set(i);}
   }
   else
   {   for(int i=0;i<ir1;i++)
         { if(Mother.b.get(i)) b.set(i);}
       for(int i=ir1;i<ir2;i++)
         { if(Father.b.get(i)) b.set(i);}
       for(int i=ir2;i<N;i++)
         { if(Mother.b.get(i)) b.set(i);}
   }
}
}


public void cross1(double crossratio,Gene Father,Gene Mother)
{// create gene from two parent genes
 // crossover process
//  Genes from Father and Mother should have the same limit
values
xmin=Father.xmin;
xmax=Father.xmax;
N=Father.N;
xN=Father.xN;
b=new BitSet(N);
int ir1=(int)(crossratio*N+0.5);
for(int i=0;i<ir1;i++)
   { if(Father.b.get(i)) b.set(i);}
for(int i=ir1;i<N;i++)
   { if(Mother.b.get(i)) b.set(i);}
}

public void cross2(double crossratio,Gene Father,Gene Mother)
{// create gene from two parent genes
 // crossover process
//  Genes from Father and Mother should have the same limit
values
xmin=Father.xmin;
xmax=Father.xmax;
N=Father.N;
xN=Father.xN;
b=new BitSet(N);
int ir1=(int)(crossratio*N+0.5);
for(int i=0;i<ir1;i++)
   { if(Mother.b.get(i)) b.set(i);}
for(int i=ir1;i<N;i++)
   { if(Father.b.get(i)) b.set(i);}
}

public long getXP()
{
//returns intermediate integer of double number equivalent

long y=1;
double r=xN;
long z=0;
for(int i=0;i<N;i++)
   {
   if(b.get(i)) {z+=y;}
   y*=2;
   }
 return z;
}

public double getX()
{//return double number equivalent of bit set number
 return (xmin+(xmax-xmin)/xN*getXP());
}

public void clear()
{//fill the entire population with zero
b.clear(0,N);}

public void fill()
{//fill the entire population with ones
b.clear(0,N);b.flip(0,N);}

public String toBitSetString()
{//return bit set array values as defined in class BitSet
return b.toString();
}
```

```
public String toString()
{
// return bit set array values as a binary number definition
// plus double value of array
 String s="(";
 for(int i=N-1;i>=0;i--)
    if(b.get(i)) s+="1";
    else         s+="0";
    s+=") "+getX();
  return s;
}

public boolean boolean_random()
{
if(Math.random()<=0.5) return false;
else return true;
}

public int int_random()
{
  int x1=(int)(Math.random()*N);
  return x1;
}

public double double_random()
{
  return (xmin+Math.random()*(xmax-xmin));
}

public void setbits()
{
b.clear();
for(int i=0;i<N;i++)
    {
   if(boolean_random()) b.set(i);
    }
}

public void flip(int i)
{
b.flip(i);
}

public void mutate()
{
  //fip one bit of data
  int i1=int_random();
  if(i1<0) i1=0;
  b.flip(i1);
}
}
```

Gene sınıfının biraz daha açılabilmesi için çeşitli örnekler verilmiştir. İlk örneğimizde 63 bitlik 5 adet Gene tipi değişken oluşturuyoruz. Değişken değerleri gelişigüzel yüklenmektedir.

Program 3-2 geneTest sınıfı ile çeşitli 63 bitlik Gen'ler oluşturma (sayı 0 ile 10 arası)

```
public class geneTest
{
  public static void main(String arg[])
   {
  Gene x1=new Gene(0,10,63);
  for(int i=0;i<5;i++)
  { x1.setbits();System.out.println(x1.toString());}
   }
}
```

Çıktı 3-3 geneTest sınıfı ile çeşitli 63 bitlik Geneler oluşturma

```
(0101010001000010001110000000010000011011000110111001011111110111)
3.291354188942094
(0110110000111110001011000010000001101001110110011010111011111110)
4.228236750730961
(1110000100011111100101101100010100000101110111001011110000010111)
8.793882590306223
(1010000100011101011110100011101000010001101010100100000110111100)
6.293560401697187
(0000000111111111110000001000000001100101000011110101111111101111)
0.078049303939407
```

Program 3-4 geneTest sınıfı ile çeşitli 63 bitlik Gen oluşturarak 3.2 değerini yükleme

```
public class geneTest
{ public static void main(String arg[])
  { Gene x1=new Gene(3.2,0,10,63);System.out.println(x1.toString());}
}
```

Çıktı 5.25.3-3 geneTest sınıfı ile çeşitli 63 bitlik Gen oluşturarak String'den 3.2 değerini yükleme

```
(0101000111101011100001010001111010111000010100011111011000000000)
3.2
```

Program 3-4 geneTest sınıfı ile çeşitli 63 bitlik Gen oluşturarak String'den değerini yükleme

```
public class geneTest
{ public static void main(String arg[])
  {Gene x1=new
Gene("0101000111101011100001010001111010111000010100011111011000000000",0,10);
    System.out.println(x1.toString());}
}
```

Çıktı 3-4 geneTest sınıfı ile çeşitli 63 bitlik Gen oluşturarak String'den değerini yükleme

```
(0101000111101011100001010001111010111000010100011111011000000000) 3.2
```

Genotype1 sınıfı gen bit-double değişken türünün bir anlamda boyutlu değişkeni gibi düşünülebilir. Genetik anlamda bakıldığında kromozona eşdeğerdir. Matematik olarak düşünüldüğünde ise n boyutlu $f(x_1,x_2,x_3,\ldots,x_n)$ fonksiyonun tüm değişkenlerine karşı gelen bir toplam değişken setidir. Fonksiyon uyum değerlerini ve kümülatif uyum değerlerini hesaplar.

Genetic algoritmanın ana sınıfı Genetic1 sınıfıdır. Bu sınıfta genetik optimizasyon gerçekleştirilmektedir. Burada bazı değişkenler kullanıcı tarafından verilmelidir. Bu değişkenler :

```
            int    POPSIZE; //populasyonu oluşturan birey sayısı
            int    MAXGENS; // nesil sayısı(maksimum
iterasyon?);
            int    NVARS;   //değişken sayısı
          int    N;       //genlerin bit eşdeğer boyutu-değişken
boyutu
          double PXOVER; //üreme-çaprazlama olasılığı;
          double PMUTATION; //mutasyon olasılığı
```

şeklindedir. Genetic sınıfı tüm Genetik optimizasyon proseslerini barındırır. Bu işlemlerden evaluate bireylerin uyumlarını (fonksiyon değerlerini) hesaplar. Genetik algoritmalarda fonksiyon değerlendirmesinde önemli olan bir konuda tüm fonksiyon değerlerinin pozitif olma gerekliliğidir. – olarak değerlendirilen fonksiyonlar sabit bir değer eklenerek artı olarak değerlendirilecek fonksiyonlara dönüştürülebilir. Bu proses fonksiyonun optimum noktasını değiştirmez.

$f(x_1,x_2,x_3,\ldots,x_n)$ fonksiyonunun maksimumu $g(x_1,x_2,x_3,\ldots,x_n) = f(x_1,x_2,x_3,\ldots,x_n) + C$ fonksiyonunun maksimumuyla aynıdır.

Bu proses kümülatif uyumu hesaplarken bir hata olmasını önler.
Evaluate prosesi bireylerin uyum değerlerini ve kümülatif uyum değerlerini hesaplar
Select metodu kümülatif uyumdaki başarılı bireyler arasından yeni nesli üretme hakkına sahip olacakları tesadüfi proses kullanarak seçer.
Crossover prosesi select prosesi tarafından seçilen bireylerden yeni bireyler üretir.
Mutate prosesi verilen belli bir yüzdedeki populasyonu mutasyona uğramasını sağlar

Calculate prosesi tüm üstteki prosesleri sırayla ve toplam nesil sayısına göre itere ederek toplam nesil sayısı sonundaki nufusu(populasyonu) elde eder.

Program 3-5 genotype1 sınıfı ve Genetic1 sınıfı

```
İmport java.io.*;
import java.text.*;
import java.util.Locale;

abstract class f_xj
{
 // single function  multi independent variable
 // a single value is returned indiced to equation_ref
 // example f[0]=x[0]+sin(x[1])
 //         f[1]=x[0]*x[0]-x[1]
 // func(x,1) returns the value of f[1]
 // func(x,0) returns the value of f[0]

 abstract double func(double x[]);
}

class genotype1
{
// each genotype1 is member of the population
// Genotype is like a chromosone in real life genetic systems
// it can be thought as an evaluation point of a function with n variable
// f(xi)=f(x1,x2,x3,..,x_nvars)
public Gene G[]      ;  // string of variables makes a genotype1
public double Fitness;  // genotype1's fitness f(xi)
public double Upper[];  // gene's upper bound; xi_lower < xi < xi_upper
public double Lower[];  // Gene's lower bound; xi_lower < xi < xi_upper
public double RFitness; // the relative fitness
public double CFitness; // cumulative fitness
public int NVARS;       // Number of variables(Genes)
public int N;           // Number of bits defining each gene
public boolean Breeder;

 public genotype1(int iNVARS,double low[],double high[],int iN,f_xj fi)
 {setgenotype1(iNVARS,low,high,iN,fi);}

 public void setgenotype1(int iNVARS,double low[],double high[],int iN,f_xj
fi)
 {
 setN(iN);
 setNVARS(iNVARS);
 G=new Gene[NVARS];
 Upper=new double[NVARS];
 Lower=new double[NVARS];
 setLower(low);
 setUpper(high);
 setGene();
 setFitness(fi);
 }

 public genotype1(genotype1 gi,f_xj fi)
 {setgenotype1(gi,fi);}

 public void setgenotype1(genotype1 gi,f_xj fi)
 {
 setN(gi.N);
 setNVARS(gi.NVARS);
 G=new Gene[NVARS];
 Upper=new double[NVARS];
 Lower=new double[NVARS];
 setLower(gi.Upper);
 setUpper(gi.Lower);
 for(int i=0;i<NVARS;i++)
   G[i]=gi.G[i].copyGene();
 setFitness(fi);
 }

 public genotype1 copygenotype(f_xj fi)
 {
 genotype1 g1=new genotype1(this,fi);
 return g1;
 }

 public void setN(int iN)    {N=iN;}
 public void setNVARS(int iNVARS)    {NVARS=iNVARS;}
 public void setLower(double lbound[]) {for(int i=0;i<NVARS;i++)
Lower[i]=lbound[i];}
 public void setUpper(double ubound[]) {for(int i=0;i<NVARS;i++)
Upper[i]=ubound[i];}
 public void setFitness(f_xj fi)
 {           Fitness=fi.func(getGene());}
 public void setRFitness(double isum)  {RFitness=Fitness/isum;}
 public void setCFitness(double ix)  {CFitness=ix;}
 public double getLower(int j) {return Lower[j];}
 public double getUpper(int j) {return Upper[j];}
 public double getFitness() {return Fitness;}
 public double getRFitness(){return RFitness;}
 public double getCFitness(){return CFitness;}
 public void breed() {Breeder=true;}
 public void not_breed() {Breeder=false;}
 public boolean isBreed() {return Breeder;}


 public void setGene(int val)
 {  G[val]=new Gene(Lower[val],Upper[val],N);}

 public void setGene()
 {  for(int j=0;j<NVARS;j++)
                setGene(j);
 }

 public double getGene(int val)
 { return G[val].getX();}

 public double[] getGene()
 {   double x[]=new double[NVARS];
               for(int j=0;j<NVARS;j++)
                 x[j]=getGene(j);
               return x;
 }

 public String toString()
 {
 //x values, Fitness,relative fitness and cumulative fitness
 String s="";
 for(int j=0;j<NVARS;j++)
           s+=G[j].toString()+" F:";
 s+=getFitness();
 s+="RF:"+getRFitness();
 s+="CF:"+getCFitness();
 return s;
 }
} //end of class genotype1

public class Genetic1
{
           // Maximizing a function
           int   POPSIZE; //population size
           int   MAXGENS; //maximum number of generations;
           int   NVARS;   //number of independent variables of function
  int   N;       //number of genes in each variable
           double PXOVER; //probability of crossover;
  double PMUTATION; //proportion of mutated variables
  int Generation; //Current generation number
  int NBreeder; //Number of survivors in a generation
  int Best; //The best genotype1 in the population
  genotype1 Population[];
  //f_xj f;  // function to be avaluated by genetic algorithm

  public Genetic1(int iPOPSIZE,int iMAXGENS,int iNVARS,int iN,
  double iPXOVER,double iPMUTATION)
  {
  POPSIZE=iPOPSIZE;
  MAXGENS=iMAXGENS;
```

```
    NVARS=iNVARS;
    N=iN;
    PXOVER=iPXOVER;
    PMUTATION=iPMUTATION;
    NBreeder=0;
    Population=new genotype1[POPSIZE+1];
            }
            public void setPopulation(double low[],double up[],f_xj fi)
            {
            for(int i=0;i<POPSIZE+1;i++)
              { Population[i]=new genotype1(NVARS,low,up,N,fi);}
            }

            public genotype1[] copyPopulation(f_xj fi)
            {
            genotype1 Po[]=new genotype1[POPSIZE+1];
            for(int i=0;i<POPSIZE+1;i++)
              { Po[i]=new genotype1(Population[i],fi);
                Po[i].Fitness=Population[i].Fitness;
                        Po[i].RFitness=Population[i].RFitness;
                Po[i].CFitness=Population[i].CFitness;
              }
            return Po;
            }

            public void evaluate(f_xj fi)
            {
              int mem;
              int j;
              double x[]=new double[NVARS];
              for(mem=0;mem<POPSIZE;mem++)
              {
              //System.out.println(Population[mem].toString());
              Population[mem].setFitness(fi);
              if(Population[mem].getFitness() >
Population[POPSIZE].getFitness())
                {
                        Best=mem;
                        Population[POPSIZE]=Population[mem];
                  for(j=0;j<NVARS;j++)
                    {

Population[POPSIZE].G[j].copyGene(Population[mem].G[j]);
                        }//end of for(j=0..
                        }//end of if
              }//end of for(mem=0;..
            }


            public void setRFCF(genotype1 Pi[])
            {
            //calculates relative and cumulative fitness functions
            int mem;
            int PN=Pi.length;
            double sum=0.0;
            //total fitness of population
            for(mem=0;mem<PN;mem++)
              {sum+=Pi[mem].Fitness;}
            //System.out.println("sum="+sum);
            //calculate relative fitness of each genotype1
            for(mem=0;mem<PN;mem++)
              {Pi[mem].setRFitness(sum);}
            //calculate cumulative fitness
            Pi[0].setCFitness(Pi[0].getRFitness());
   for(mem=1;mem<PN;mem++)
              {Pi[mem].setCFitness(Pi[mem-
1].getCFitness()+Pi[mem].getRFitness());
              }
            }

            public void toString(genotype1 Pi[])
            {
            int mem;
            int PN=Pi.length;
            for(mem=0;mem<PN;mem++)
            {
              //list them
              System.out.println("Population["+mem+"]="+Pi[mem].toString());
              }
              }

              public void select(f_xj fi)
              {
              //select the new generation members of population
              double r;
              int mem;
              setRFCF(Population);
              //create a new population;
              genotype1 Po[]=copyPopulation(fi);
    setRFCF(Po);
    for(int i=0;i<POPSIZE;i++)
    {   mem=0;
      r=Math.random();
      double gf1=Population[i].getCFitness();
                double gf2;
      if (gf1 > r) Po[i]=Population[mem].copygenotype(fi);
                for(mem=1;mem<POPSIZE;mem++)
        {
                  gf2=Population[mem].getCFitness();
                  if( gf2>=r && gf1< r )
{Po[i]=Population[mem].copygenotype(fi);break;}
        }
    }
    setRFCF(Po);
              Population=Po;
              evaluate(fi);
    //toString(Population);
    }

    public void crossover(f_xj fi)
    {
    int i,count;
    int mem;
    int POne,a;
    int PTwo,b;
    int point;
    //select members for breeding
    int iselect[]=new int[POPSIZE];
    int counter=0;
    double r;
     for(mem=0;mem<POPSIZE;mem++)
              {
               r=Math.random();
               if(r < PXOVER)
                 {
                        iselect[counter++]=mem;
                 Population[mem].breed();
                        NBreeder++;
                 }
               else
                 Population[mem].not_breed();
              }
              //System.out.println("iselect="+Matrix.toString(iselect));
              //let also best of the population to breed
              Population[Best].breed();

    //loop through the population select breeding pairs
     for(mem=0;mem<POPSIZE;mem++)
      {
        //select two in popolation in random
        a=(int)(Math.random()*NBreeder)+1;
        b=(int)(Math.random()*NBreeder)+1;
        count=0;
        POne=0;
        PTwo=0;
        //select two individuals for breeding
        for(i=0;i<POPSIZE;i++)
          {
          if(Population[i].isBreed())
            {
            count++;
            if(count==a) POne=count;
```

```
            if(count==b) PTwo=count;
                        }
                      }
                    //perform a crossover;
                      genotype1 Po[]=new genotype1[2];
                              Po[0]=Population[POne].copygenotype(fi);
                              Po[1]=Population[PTwo].copygenotype(fi);
                      for(i=0;i<NVARS;i++)
                      {

Population[POne].G[i].cross1(Math.random(),Po[0].G[i],Po[1].G[i]);

Population[PTwo].G[i].cross1(Math.random(),Po[0].G[i],Po[1].G[i]);
                              }
                              Population[POne].setFitness(fi);
                              Population[PTwo].setFitness(fi);
                    }
                  }

  public void mutate(f_xj fi)
  {
  int i;
  double lbound,hbound;
  int nmutations;
  int member;
  int var;
  nmutations=(int)((double)(POPSIZE*NVARS*PMUTATION*N));
  for(i=0;i<nmutations;i++)
    {
    member=(int)(Math.random()*POPSIZE);
    var=(int)(Math.random()*NVARS);
    //replace the old value with a new mutated one
    Population[member].G[var].mutate();
              //after mutation recalculate the function value
    Population[member].setFitness(fi);
    }
  }

  public String report()
  {
  String s;
  int i;
  double best_value;
  double avg;
  double stdev;
  double sum_square;
  double square_sum;
  double sum;
  double xx;
  sum=0;
  sum_square=0.0;
  for(i=0;i<POPSIZE;i++)
    {
              xx= Population[i].getFitness();
    sum+=xx;
    sum_square=xx*xx;
              }
  avg=sum/(double)POPSIZE;
  square_sum=sum*sum/(double)POPSIZE;
  stdev=Math.sqrt(square_sum);
  best_value=Population[POPSIZE].getFitness();
  double aa[]=new double[NVARS+1];
  for(i=0;i < NVARS;i++)
    {
   aa[i]=Population[POPSIZE].getGene(i);
              }
              aa[NVARS]=Population[POPSIZE].getFitness();

  //s="Generation = "+Generation+"best value =
"+Matrix.toString(aa)+"Average = "+avg+"Standart Deviation = "+stdev;
  s="Generation = "+Generation+"best value = "+Matrix.toString(aa);
  return s;
  }

  public double[] getBest()
  {
  double aa[]=new double[NVARS+1];
  for(int i=0;i < NVARS;i++)
    {
   aa[i]=Population[POPSIZE].getGene(i);
              }
              aa[NVARS]=Population[POPSIZE].getFitness();
  return aa;
  }

  public double[] calculate(f_xj fi,boolean report)
  {
  evaluate(fi);
  Generation=0;
  while(Generation<MAXGENS)
    {
    Generation++;
    NBreeder=(int)(0.8*POPSIZE);
    select(fi);
    crossover(fi);
    mutate(fi);
    evaluate(fi);
    //if(report)
      //System.out.println(report());
    }
  return getBest();
  }
}
```

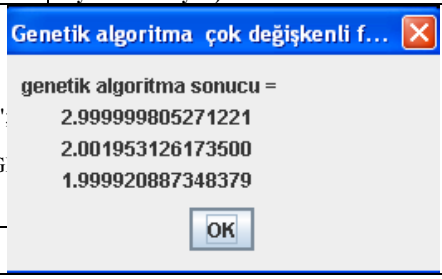
örnek fonksiyon olarak $f(x_0, x_1)= 1.0+\text{Math.cos(Math.PI*(}\ x_0\text{-3.0))*Math.cos(2.0*Math.PI*(}\ x_1\text{-2.0))/ (1+(}\ x_0\text{-3.0)*(}\ x_0\text{-3.0)+(}\ x_1\text{-2.0)*(}\ x_1\text{-2.0));}$ fonksiyonuna bakalım. Bu fonksiyonun x_0 -10 ile 10 ve x_1 -10 ile 10 arasındaki maksimum değerini bulmak istiyoruz.

Problem 5.15.5-5 OPO19C örnek problem sınıfı (iki bilinmeyenli fonksiyon)

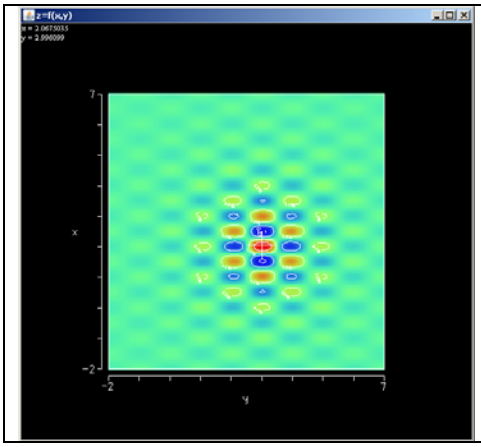
```
İmport java.io.*;
import javax.swing.*;


class fa extends f_xj
{
//çözümü istenen fonksiyon
 double func(double x[])
 {
   double ff= 1.0+Math.cos(Math.PI*(x[0]-3.0))*Math.cos(2.0*Math.PI*(x[1]-
2.0))/
   (1+(x[0]-3.0)*(x[0]-3.0)+(x[1]-2.0)*(x[1]-2.0));
   return ff; //maksimum testi;
  }
}

class fb extends f_xj
{
public double func(double x[])
{
//çözümü istenen fonksiyon
double ff;
fa fx=new fa();
return -fx.func(x); //maksimum testi
}
}

class OPO19C
{
   public static void main(String args[]) throws IOException
   {
   int N=63;
              int POPSIZE=200;
   int MAXGENS=200;
   int NVARS=2;
   double PXOVER=0.3;
   double PMUTATION=0.02;
   fa fax=new fa();
   fb fbx=new fb();
```

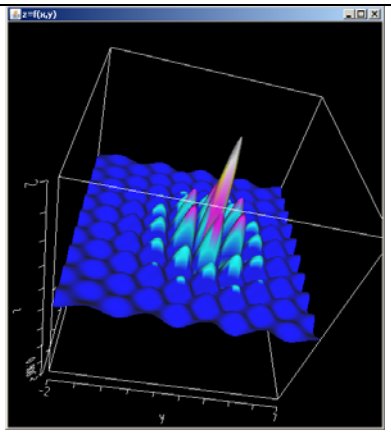
```
    Genetic1 xx=new
Genetic1(POPSIZE,MAXGENS,NVARS,N,PXOVER,PMUTATION);
    double low[]=new double[NVARS];
    double high[]=new double[NVARS];
    low[0]=-10.0;high[0]=10.0;
    low[1]=-10.0;high[1]=10.0;
    xx.setPopulation(low,high,fax);
    double a[]=xx.calculate(fax,false);
    double x0[]=new double[NVARS];
    for(int i=0;i<NVARS;i++) x0[i]=a[i];
    String s="genetik algoritma sonucu =\n"+Matrix.toStringT(a);
     String s2="Genetik algoritma  çok değişkenli fonksiyon optimizasyonu ";

JOptionPane.showMessageDialog(null,s,s2,JOptionPane.PLAIN_MESSAG
    }
}
```

Çıktı 5.15.5-8 OPO19C örnek problem sınıfı (iki bilinmeyenli fonksiyon)



Çıktı 5.15.5-9 OPO19C örnek problem sınıfı (iki bilinmeyenli fonksiyon) üç boyutlu grafik çıktısı





4.SONUÇLAR

Optimizasyon matematik ve mühendislik açısından oldukça önemli bir bilim dalıdır. Genetik algoritmalarla optimizasyon ise günümüzde bir çok alanda, bilhassa değişken sayısının çok olduğu ve geometrik yöntemlerin kompleks fonksiyon yapılarından dolayı iyi bir şekilde kullanılamadığı durumlarda oldukça yararlıdırlar.