ANKARA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

OPTİMİZASYONDA GENETİK ALGORİTMA YAKLAŞIMI

Elif BAĞCIOĞLU

MATEMATİK ANABİLİM DALI

ANKARA
2021

ÖZET

Yüksek Lisans Tezi

OPTİMİZASYONDA GENETİK ALGORİTMA YAKLAŞIMI

Elif BAĞCIOĞLU

Ankara Üniversitesi
Fen Bilimleri Enstitüsü
Matematik Anabilim Dalı

Danışman: Doç. Dr. Arzu ÜNAL

Bu tez dört bölümden oluşmaktadır. Birinci bölümde optimizasyon ve optimizasyon sınıflandırmaları hakkında temel kavramlar verilmiştir. İkinci bölümde, evrim ve genetik hakkında tarihsel ve kavramsal temel bilgiler verilmiş ve bunların Genetik Algoritmalar ile olan ilişkileri ayrıntılı bir biçimde irdelenmiştir. Genetik Algoritmalar yöntemsel olarak bu bölümde verilmiştir. Genetik Algoritmaların farklı matematik problemleri üzerindeki örneklerine de Üçüncü bölümde değinilmiştir. Son bölümde sonuçlar özetlenmiştir.

Temmuz 2021, 54 sayfa

Anahtar Kelimeler: Genetik algoritma, optimizasyon, evrimsel hesaplama, evrim.

ABSTRACT

Master Thesis

GENETIC ALGORITHM APPROACH IN OPTIMIZATION

Elif BAĞCIOĞLU

Ankara University
Graduate School of Natural and Applied Sciences
Department of Mathematics

Supervisor: Doç. Dr. Arzu ÜNAL

This thesis consists of four chapters. In the first chapter, basic concepts about optimization and optimization classifications are given. In the second chapter, historical and conceptual basic information about evolution and genetics is given and their relations with Genetic Algorithms are examined in detail. Genetic Algorithms are methodically given in this section. Examples of Genetic Algorithms on different mathematical problems are also mentioned in the third chapter. In the last section, the results are summarized.

July 2021, 54 pages

Keywords: Genetic algorithm, optimization, evolutionary computation, evolution.

TEŞEKKÜR

Çalışmam boyunca beni yönlendiren, araştırmalarımın her aşamasında bilgi, öneri ve yardımlarını benden esirgemeyen değerli danışmanım Sayın Doç. Dr. Arzu ÜNAL'a ve bugüne kadar emeği olan bütün hocalarıma ve çalışmalarım boyunca sabır gösteren ve desteğini esirgemeyen canım Aileme teşekkür ederim.

Elif BAĞCIOĞLU
Ankara, Temmuz 2021

İÇİNDEKİLER

SİMGELER DİZİNİ

*	Çarpım İşareti
P(c)	Kromozom Olasılığı
P(m)	Mutasyon Olasılığı
F(x)	Fonksiyon
F_obj	Amaç Fonksiyonu
Fitness	Uygunluk Değeri
P[i]	i. Olasılık
C[i]	i. Birikimli Olasılık
R[i]	i. Rastgele Değer
ρm	Mutasyon Oranı
ρc	Çaprazlama Oranı
V	Volt

Kısaltmalar

GA	Genetik Algoritma

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

1. GİRİŞ

Günlük hayatımızın birçok alanında optimizasyonla karşı karşıya kalırız. Uyku miktarını en üst düzeye çıkarırken yine de işimizi yetiştirebilmek için sabah kaçta kalkılacağından, işe gitmek için en iyi güzergahın belirlenmesine, hangi proje ile başlanacağından, en az malzemeyle en çok kişiyi doyuracak yemeğin hazırlanmasına kadar hayatımızın birçok anı optimizasyon ile iç içedir. Bir ürün tasarlanırken, bu ürünün maliyetini en aza indirmek veya cazibesini en üst düzeye çıkarmak için, bu ürünün uzunluğunu kısaltmak veya ağırlığını azaltmak gerekebilir. Dolayısıyla optimizasyon bazı şeyleri "daha iyi" yapmakla ilgilidir denebilir. Bu, bir şirketin kazancını en üst düzeye çıkarmak için daha iyi kararlar vermesi anlamına gelebileceği gibi; bir fabrikanın daha az çevresel etkiye sahip ürünler üretmesini sağlaması ya da bir zooloğun bir hayvanın diyetini iyileştirmesine yardım etmesi anlamına gelebilir. Optimizasyondan bahsedildiğinde sıklıkla "en iyi" veya "iyileştirme" gibi terimler kullanılır (Özturan 2019). En iyi'nin bazı durumlarda fazlalığı (örneğin kâr) bazı durumlarda da azlığı (örneğin çöp-atık veya maliyet) temsil edebildiğine dikkat edilmelidir. Başka bir ifadeyle en iyi kazançta kazancın maksimum olması beklenirken, en iyi atıkta atığın minimum olması hedeflenir.

Diğer taraftan "en iyi" çözüm terminolojisi, birden fazla çözüm olduğunu ve çözümlerin eşit olmadığını ifade eder. Bazı problemlerin kesin cevapları veya kökleri vardır (matematiksel anlamda) ve dolayısıyla en iyinin tanımı belirlidir. Bu duruma örnek olarak pokerdeki en iyi el veya birinci basamaktan doğrusal bir diferansiyel denklemin çözümü verilebilir. Diğer problemler, optimal noktalar veya ekstremumlar olarak bilinen çeşitli minimum veya maksimum çözümlere sahiptir ve buradaki "en iyi" göreceli olarak değişebilir. Örnekler içinde en iyi sanat eseri veya en iyi müzik bestesi verilebilir. Bu durumda en iyi'nin tanımı, eldeki probleme, çözüm yöntemine ve izin verilen toleranslara bağlıdır. Dolayısıyla en uygun çözüm, problemi formüle eden kişiye bağlıdır denebilir (Haupt 2004).

Optimizasyon, minimum veya maksimum çıktı (veya sonucu) bulmak için girdileri veya bir cihazın özelliklerini, matematiksel süreci veya deneyi ayarlama işlemidir. Bir

optimizasyon problemi, kesin matematiksel terimlerle ifade edilmeli, tam olarak çözülmek istenen olayı yansıtmalıdır. Optimizasyon problemleriyle uğraşırken, ele alınması gereken iki önemli adım vardır. İlk olarak, amaç veya maliyet fonksiyonu, optimize edilecek problemi hassas bir biçimde karşılamalıdır. Daha arzu edilen bir çözümde, daha küçük bir maliyet fonksiyonu, minimum süre, daha yüksek verimlilik, daha fazla fayda, minimum kayıp vb. olgular söz konusu olmalıdır. Maliyet fonksiyonu optimizasyon kriterini doğru şekilde yansıtmıyorsa, nihai çözüm muhtemelen aranan optimum çözüm olmayacaktır. İkinci olarak, makul çözümlerin ele alınan problemde gerçekten uygulanabilir olması için göz önüne alınması gereken kısıtlamaları açıkça belirtmek de aynı derecede önemlidir. Bu kısıtlamalar doğru bir şekilde yazılmazsa, bazıları göz ardı edilirse veya çok kısıtlayıcı olan birkaçı uygulanırsa, bir kez daha, elde edilecek çözüm, gerçek çözüm olmayabilir (Pedregal 2004).

1.1 Optimizasyon Örnekleri

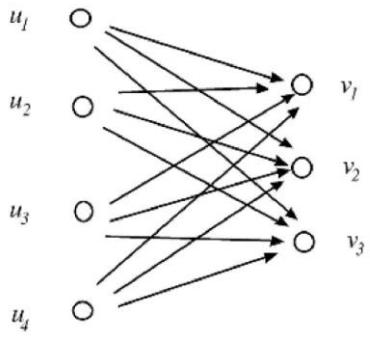
Bu kısımda bazı iyi bilinen optimizasyon örnekleri verilmektedir.

Örnek 1.1 Ulaşım Problemi: Belirli bir ürün, n servis noktasından $v_1, v_2, \dots, v_m$ miktarlarında alınacağı m varış noktasına $u_1, u_2, \dots, u_n$ miktarlarında sevk edilecektir. (Bakınız Şekil 1.1) Bir birim ürünün i çıkışından j varış noktasına gönderilme maliyetinin c_{ij} olduğu bilindiğine göre, toplam nakliye maliyeti minimum olacak şekilde, i çıkışından j varış noktasına gönderilecek x_{ij} üretim miktarının belirlenmesi problemini ele alalım.

Bu durumda toplam maliyet miktarı $\sum_{i,j} c_{ij}x_{ij}$ dir. Şimdi problemin kısıtlarını inceleyelim:

Sabit bir i çıkış noktasından sevk edilecek miktar u_i ise

$$\sum_j x_{ij} = u_i\,, \quad i = 1, \dots, n$$



Şekil 1.1 Bir ulaşım ağı diyagramı

ve benzer olarak her sabit j varış noktası için geri dönecek miktar v_j için,

$$\sum_i x_{ij} = v_j\,, \qquad j = 1, \dots, m$$

sağlanmalıdır. Bu eşitliklerin

$$\sum_i u_i = \sum_j v_j$$

eşitliğinin sağlanması durumunda uyumlu olduğuna dikkat edilmelidir. Bu aynı zamanda problemin iyi kurulması için verilerin karşılaması gereken bir kısıtlamadır. Ayrıca bir servis noktası veya bir varış noktası olma özelliklerinin yer değiştirmeyeceği kabul ediliyorsa $\forall i, j$ için $x_{ij} > 0$ olmalıdır. Böylece optimizasyon problemi;

$$Min \sum_{i,j} c_{ij} x_{ij}$$

$$\sum_j x_{ij} = u_i\,, \quad i = 1, \dots, n$$

$$\sum_i x_{ij} = v_j, \quad j = 1, \dots, m$$

$$x_{ij} > 0, \quad \forall i, j$$

matematiksel formuna indirgenir.

Örnek 1. 2 Diyet Problemi: Belirli yiyeceklerin besleyici içerikleri, fiyatları ve her besin için gereken günlük minimum miktarlar bilinmektedir. Amaç, her besin için gerekli olan minimum miktar karşılanacak şekilde ve diyetin toplam maliyeti mümkün olduğunca düşük olacak biçimde, satın alınması gereken yiyeceklerin miktarlarının belirlenmesidir. c_i i gıdasının birim fiyatı ve x_i de i gıdasından alınması gereken miktar ise, minimize edilmek istenen toplam maliyet;

$$\sum_i c_i x_i$$

biçimindedir. a_{ij}, i gıdasının birim başına j besin içeriği ve b_j de besin maddesinin günlük minimum gereksinimi olsun. Bu durumda diyet seçiminde karşılanması gereken minimum;

$$\sum_i a_{ij} x_i \geq b_j \,, \quad \forall\, j$$

olup son olarak negatif olmama koşulu $x_i \geq 0, \forall i$ ile diyet problemi;

$$min \sum_i c_i x_i$$

$$\sum_i a_{ij} x_i \geq b_j \,, \quad \forall\, j$$

$$x_i \geq 0, \quad \forall i$$

biçiminde ifade edilir.

1.2 Optimizasyon Sınıflandırmaları

Şekil 1.2'de, optimizasyon algoritmaları altı kategoriye ayrılmaktadır. Bu altı kategorinin ya da alt dalların hiçbiri birbirini dışlamaz. Başka bir deyişle, bir dinamik optimizasyon problemi kısıtlı veya kısıtsız olabilir. Ayrıca bazı değişkenlerin bazıları ayrık, bazıları ise sürekli olabilir.

Şekil 1.2 Bir optimizasyon sınıflandırması diyagramı

Üstten başlayarak saat yönünde bu sınıflandırmaları inceleyelim;

- Deneme-yanılma optimizasyonu, çıktıyı üreten süreç hakkında fazla bir şey bilmeden çıktıyı etkileyen değişkenleri ayarlama sürecini ifade eder. Basit bir örnek, en iyi görüntü ve ses alımını elde etmek için bir TV'deki tavşan kulaklarını ayarlamaktır. Bir anten mühendisi, tavşan kulaklarının belirli bükülmelerinin neden diğer bükülmelerden daha iyi bir görüntü ile sonuçlandığını yalnızca tahmin edebilir. Deneyciler bu yaklaşımı tercih ederler. Penisilinin bir antibiyotik olarak keşfedilmesi ve iyileştirilmesi gibi birçok büyük keşif, optimizasyona yönelik deneme yanılma yaklaşımından kaynaklanmıştır. Buna karşılık, matematiksel bir formül, fonksiyon optimizasyonunda amaç fonksiyonunu tanımlar. Fonksiyonun çeşitli matematiksel manipülasyonları, optimal çözüme yol açar. Teorisyenler bu teorik yaklaşımı tercih ederler.

- Yalnızca bir değişken varsa, optimizasyon tek boyutludur. Birden fazla değişkeni olan bir problem çok boyutlu optimizasyon gerektirir. Boyut sayısı arttıkça optimizasyon giderek zorlaşır. Birçok çok boyutlu optimizasyon yaklaşımı, bir dizi tek boyutlu yaklaşıma genelleştirilir.

- Dinamik optimizasyon, çıktının zamanın bir fonksiyonu olduğu, statik ise çıktının zamandan bağımsız olduğu anlamına gelir. Örneğin işe gidip gelmenin birkaç yolu vardır. Peki en iyi yol nedir? Uzak bir bakış açısından, sorun statiktir ve çözüm bir harita veya bir arabanın kilometre sayacı kullanılarak bulunabilir. Pratikte, rotalardaki sayısız varyasyon nedeniyle bu problem basit değildir. En

kısa rota mutlaka en hızlı rota değildir. En hızlı rotayı bulma problemi, çözümü günün saatine, hava durumuna, kazalara vb. bağlı olan dinamik bir problemdir. Statik problemin en iyi çözüm için çözülmesi zordur, ancak zamanın eklenen boyutu dinamik problemi çözme zorluğunu daha da arttırır.

- Optimizasyon, ayrık veya sürekli değişkenlerle de ayırt edilebilir. Kesikli değişkenler yalnızca sınırlı sayıda olası değere sahipken, sürekli değişkenler sonsuz sayıda olası değere sahiptir. Bir listedeki bir dizi göreve hangi sırayla başlayacağımıza karar veriyorsak, ayrık optimizasyon kullanılır. Ayrık değişken optimizasyonu, kombinatoryal optimizasyon olarak da bilinir, çünkü optimum çözüm, tüm olası değişkenlerin sonlu havuzundan belirli bir değişken kombinasyonundan oluşur. Ancak bir sayı doğrusu üzerinde f (x)'in minimum değerini bulmaya çalışıyorsak, problemi sürekli olarak görmek daha uygundur.
- Değişkenlerin genellikle sınırları veya kısıtlamaları vardır. Kısıtlı optimizasyon, değişken eşitlikleri ve eşitsizlikleri maliyet fonksiyonuna dahil eder. Kısıtlanmamış optimizasyon, değişkenlerin herhangi bir değer almasına izin verir. Kısıtlı bir değişken, genellikle değişkenlerin dönüştürülmesi yoluyla sınırlandırılmamış bir değişkene dönüşür. Sayısal optimizasyon rutinlerinin çoğu en iyi şekilde sınırlandırılmamış değişkenlerle çalışır. $-1 \leq x \leq 1$ aralığında $f(x)$'in minimumunu arayan basit bir kısıtlı optimizasyon örneğini ele alalım. $x = sin(u)$ dönüşümü ile x değişkeni için kısıtlı problem, u değişkeni için $f(sin(u))$'yi minimize eden kısıtsız bir probleme indirgenmiş olur. Kısıtlı optimizasyon problemi, değişkenleri doğrusal denklemler ve doğrusal kısıtlamalar cinsinden formüle ediliyorsa, buna doğrusal programlama problemi adı verilir. Maliyet denklemleri (fonksiyonları) veya kısıtlamaları doğrusal olmadığında, problem doğrusal olmayan bir programlama problemi haline gelir.
- Bazı algoritmalar, değişken değerlerinin bir başlangıç değer kümesinden başlayarak maliyeti en aza indirmeye çalışır. Bu minimum arayanlar, yerel minimumlarda kolayca takılıp kalırlar ancak hızlı olma eğilimindedirler. Bunlar geleneksel optimizasyon algoritmalarıdır ve genellikle hesap yöntemlerine dayanırlar. Bir değişken kümesinden diğerine geçiş, bazı belirleyici adımlar dizisine dayanır. Öte yandan, rastgele yöntemler, değişken kümeleri bulmak için

bazı olasılık hesaplamaları kullanır. Daha yavaş olma eğilimindedirler, ancak genel minimumu bulmada daha başarılıdırlar.

Genetik algoritma (GA), doğal evrimden esinlenen sezgisel bir arama ve optimizasyon tekniğidir. GA'lar ilk olarak John Holland (1975) tarafından hesaplama açısından zor olan problemlere optimal çözümler bulmak için bir araç olarak önerildi. Holland'ın Şema Teoremi ve ilgili blok hipotezi, verimli GA'ların tasarımı için teorik ve kavramsal bir temel sağlamıştır. Ayrıca, son derece modüler yapıları nedeniyle GA'ları uygulamanın kolay olduğu kanıtlanmıştır. Sonuç olarak, alan hızla büyümüş ve fen bilimleri, mühendislik, ekonomi, ekoloji, insan bağışıklık sistemi, popülasyon genetiği ve endüstrideki önemli karmaşıklıkta çok çeşitli gerçek dünya problemlerine başarıyla uygulanmıştır. (Satman 2016, McCall 2005, Emel ve Taşkın 2002, Mateescu 2006). GA teorisi, kendisinden önceki teorilerin öngöremediği fenomenleri tanımlamak ve açıklamak için kullanılan bir dizi yaklaşımla aktif ve büyüyen bir alandır. Bu tezde Genetik Algoritmaların metodolojisi ayrıntılı biçimde incelenecek ve çeşitli optimizasyon problemlerine nasıl uygulandığına değinilecektir.

2. GENETİK ALGORİTMA

2.1 Genetik Algoritmalar: Tarihçe ve Teori

1960'ların başında birbirinden bağımsız olarak bilim insanları evrimin mühendislik ve matematik problemleri için bir optimizasyon aracı olarak kullanılabileceği fikrini ortaya attı. Temel fikir, doğal genetik varyasyondan ve doğal seçilimden esinlenen operatörleri kullanarak belirli bir probleme aday çözümlerden oluşan bir popülasyon geliştirmekti. Darwin'in Evrim teorisinden esinlenen ve genel olarak Evrimsel Hesaplama olarak adlandırılan bu optimizasyon yöntemlerinden Genetik Algoritma, Holland tarafından 1960'ların başında (1975), Evrim Stratejileri Rechenberg (1973) ve Schwefel, Evrimsel Programlama Fogel ve arkadaşları (1966) ve Genetik Programlama da Koza (1992) tarafından geliştirilmiştir (Gen ve Cheng 1997, M. Gen and R. Cheng 1999).

Holland, Michigan Üniversitesi'nde psikoloji, bilgisayar bilimi ve elektrik mühendisliği profesörüydü. Sistemlerin "çevrelerine nasıl uyum sağladığını" anlama çabası onu harekete geçirdi ve Genetik algoritmaları (GA'lar) ilk olarak 1960'larda ortaya koydu. Öğrencileri ve meslektaşları ile yıllarca süren çalışmalardan sonra, 1975'te Doğal ve Yapay Sistemlerde Adaptasyon adlı ünlü kitabını yayınladı (Holland 1975). Bu kitapta Holland, genetik algoritmaları "biyolojik evrimin bir soyutlaması olarak sundu ve genetik algoritmalar altında adaptasyon için teorik bir çerçeve" verdi. Bu kitap, hesaplamalı evrim için teorik bir temel öneren ilk kitaptı ve yakın zamana kadar genetik algoritmalar üzerindeki tüm teorik çalışmaların temeli olarak kaldı. Evrimin arkasındaki matematiği göstermesi nedeniyle bir klasik haline geldi (Simon 2013, Mitchell 1996).

Evrim stratejilerinin ve evrimsel programlamanın aksine, Holland'ın asıl amacı belirli problemleri çözmek için algoritmalar tasarlamak değil, doğada meydana geldiği şekliyle adaptasyon olgusunu resmi olarak incelemek ve doğal adaptasyon mekanizmalarını bilgisayar sistemlerine adapte edilebileceği yollar geliştirmekti (Mitchell 1996).

GA'lar, güçlü ve geniş çapta uygulanabilir stokastik arama ve optimizasyon teknikleri olarak, günümüzde belki de en yaygın olarak bilinen evrimsel hesaplama yöntemlerinden birisidir. GA'ın avantajlarından bazıları şunlardır:

• Sürekli veya ayrık değişkenlerle optimize eder,

• Türev bilgisi gerektirmez,

• Maliyet yüzeyinin geniş bir örnekleminden eş zamanlı olarak arama yapar,

• Çok sayıda değişkenle ilgilenir,

• Paralel bilgisayarlar için çok uygundur,

• Son derece karmaşık maliyet yüzeylerine sahip değişkenleri optimize eder (yerel minimumdan atlayabilirler),

• Tek bir çözüm değil, optimum değişkenlerin bir listesini sağlar,

• Optimizasyonun kodlanmış değişkenlerle yapılması için değişkenleri kodlayabilir ve

• Sayısal olarak oluşturulmuş verilerle, deneysel verilerle veya analitik fonksiyonlarla çalışır.

Bu avantajlar ilgi çekicidir ve geleneksel optimizasyon yaklaşımları bir şekilde başarısız olduğunda çarpıcı sonuçlar üretir. Elbette, GA her sorunu çözmenin en iyi yolu değildir. Örneğin, geleneksel yöntemler, yalnızca birkaç değişkenli iyi tanımlı bir dışbükey analitik fonksiyonun çözümünü hızla bulmak için daha uygundurlar. Bu gibi durumlarda, analize dayalı yöntemler GA'dan daha iyi performans gösterir ve GA hala ilk popülasyonun maliyetlerini analiz ederken, geleneksel yöntem minimumu hızla bulur. Bu problemler için optimizasyonun teorik deneyimini kullanmalı ve bu hızlı yöntemler tercih edilmelidir. Ancak, birçok gerçekçi problem bu kategoriye girmez. Ayrıca aşırı zor olmayan problemler için diğer yöntemler çözümü GA'dan daha hızlı bulabilir. GA'ya gücünü veren geniş çözüm popülasyonu, seri bir bilgisayarda hız söz konusu olduğunda dezavantaja dönüşür, çünkü bu çözümlerin her birinin maliyet fonksiyonu hesaplanmalıdır. Ancak paralel bir bilgisayar varsa, her işlemci aynı anda ayrı bir fonksiyonu değerlendirebilir. Böylece GA, bu tür paralel hesaplamalar için optimal olarak uygundur.

Bir genetik algoritma (herhangi bir evrim programı gibi) belirli bir problem için Michalewicz (1992) tarafından özetlenen aşağıdaki beş temel bileşene sahip olmalıdır:

- potansiyel çözümler için probleme genetik bir temsil,
- potansiyel çözümlerden oluşan bir başlangıç popülasyonu oluşturmanın bir yolu,
- çevrenin rolünü oynayan bir değerlendirme fonksiyonu, çözümleri "uygunluk" açısından derecelendirme,
- yavruların (döllerin) kompozisyonunu değiştiren genetik operatörler,
- genetik algoritmanın kullandığı çeşitli parametreler için değerler (popülasyon büyüklüğü, uygulanan genetik operatörlerinin olasılıkları, vb.).

Genetik algoritma, t kuşağı için P(t) gibi bireylerin bir popülasyonunu sağlar. Her birey, eldeki probleme potansiyel bir çözümü temsil eder. Her birey, uygunluğunun bir ölçüsünü vermek için değerlendirilir. Bazı bireyler, yeni bireyler oluşturmak için genetik işlemler aracılığıyla stokastik dönüşümlere uğrarlar. İki tür dönüşüm söz konusudur; tek bir bireydeki değişiklikle yeni bireyler oluşturan *mutasyon* ve iki bireyin parçalarının birleşmesiyle yeni bireyler oluşturan *çaprazlama.* Böylece döl denilen yeni C(t) bireyleri üretilmiş olur. Yeni bir popülasyon aile ve döl popülasyonundan daha uygun bireyler seçilerek oluşturulur. Çeşitli jenerasyonların ardından algoritma, problemin optimal çözümünü temsil etmesi umulan en iyi bireye yakınsar. Genetik algoritmanın genel yapısı aşağıdaki algoritmadaki gibidir (Gen ve Cheng 1999):

```
begin
 t ← 0;
 initialize P(t);
 evaluate P(t);
 while (not termination condition) do
 begin
  recombine P(t) to yield C(t);
  evaluate C(t);
  select P(t + 1) from P(t) and C(t);
  t ← t + 1;
 end
end
```

Şekil 2.1 Genel Bir Genetik Algoritma Yapısı

Genetik algoritma yöntemini daha ayrıntılı vermeden önce, genetiğin ve evrimin tarihine bir göz atmakta fayda vardır.

2.2 Evrim ve Genetik: Tarihçe ve Teori

Genetik çalışmaları Gregor Mendel'in (1822-1884) deneyleriyle başlamıştır. Genetiğin babası olarak bilinen Gregor Mendel, kalıtımın genetik tabanı üzerinde çalışmıştır. Bu çalışmalara göre, özellikler biyolojik objeler olan genler tarafından kontrol edilmektedir. Bu genler çok çeşitli yollarla üreme hücreleri olan yumurta ve spermlerin üretimi sırasında ayrılmaktadır.

Mendel kalıtım sistemini formülize etmiştir. Özelliklerin dağılımını dikkatli bir şekilde gözlemleyerek, ilk yasası olan ayrım ilkesini vermiştir. Bu yasaya göre her bir ebeveynden birer tane olmak üzere çiftler halinde miras alınan faktörler olmalıdır. Mendel'in ikinci yasası, bağımsız çeşitlilik ilkesidir. Bu ilke, bir özellik için alel kalıtımının bir diğer özellik için olandan bağımsız olduğunu belirtir. Örneğin, bireyin boyu ile göz renginin belirlenmesi birbirinden bağımsızdır. (Haupt ve Haupt 2004)

1858 yılında Charles Darwin (1809-1882) ve Alfred Russel Wallace (1823-1913) bağımsız olarak evrimin olduğuna dair kanıtlar vermiş ve evrimin nasıl oluştuğuna dair mekanizmalar öne sürmüşlerdir. Evrim kuramının babası olarak tanımlanan Darwin 1859 yılında "Türlerin Kökeni (The Origin of Species)" ve 1871 yılında "İnsanın Türeyişi (Descent of Man)" adlı yapıtlarını yayınlamış ve tüm yaşam formlarının (insan da dahil olmak üzere) tek bir ortak kökenden geldiklerini savunmuştur. Darwin'in evrim teorisi dört temel önermeye dayanmaktadır. Birincisi, benzer benzeri doğurur; başka bir ifadeyle, bir yavru, ebeveynlerinin birçok özelliğine sahiptir. Bu öncül, popülasyonun istikrarlı olduğunu ima eder. İkincisi, bireyler arasında bir nesilden diğerine aktarılabilen özelliklerde farklılıklar olduğudur. Üçüncü öncül, üretilen yavruların yalnızca küçük bir yüzdesinin yetişkinliğe kadar hayatta kalacağıdır. Son olarak, yavrulardan hangisinin hayatta kalacağı, onların kalıtsal özelliklerine bağlıdır. Bu öncüller birleşerek, doğal seçilim (natural selection) teorisini üretirler. Modern evrim teorisinde genetik anlayışı, doğal seçilimin aşamalarının açıklanmasına hız kazandırır.

Darwin'in gözlemleri tüm organizmaların geometrik olarak artan oranlarla çoğaldıkları, ancak türlerdeki sayının aşağı yukarı sabit olduğunu göstermiştir. Buradan devam eden savaşım sonucunda türlerin varoluşu ve hayatta kalma mücadelesi ortaya çıkmaktadır. Aynı tür içindeki farklı varyasyonların ortaya çıkışı, hayvan ya da bitki türlerinin ortaya çıkışında ve hayatta kalma sürecinde ve üremelerinde daha avantajlı olmalarına yardım etmektedir. Bu olumlu varyasyonlar oğul döllere aktarılmakta ve yeni jenerasyonlar birbiri ardını izleyen başarılı türleri yaygınlaştırmaktadır. Bu sonuç "doğal seçilim prensibi" olarak adlandırılmaktadır. Aynı yolla, seksüel seçilim de önemli bir rol oynamaktadır. Bu teorinin gelişimiyle aşamalı yığılmalarla bireysel modifikasyon ile türlerin kökeni ve çeşitliliği ortaya konulmaktadır.

R.A. Fisher ve J.B.S. Haldane, Darwin'in doğal seçilim teorisiyle, Mendel'in genetik teorisini birleştirip "Modern Sentez" ya da "Neo-Darwinizm" olarak isimlendirmişlerdir. Başka bir deyişle Neo-darwinizm, bu iki teorinin bir kombinasyonudur ve böylece Darwin ile Wallace'in genetik eksiği tamamlanmış ve Mendel genetiğinin de evrim teorisine integrasyonu sağlanmıştır. Modern sentezin ana katkısı, kalıtımın ve dolayısıyla evrimin temel birimi olan genler ile, yeni edinilen bilgilerle evrimin mekanizması; yani doğal seçilim arasındaki bağlantıyı kurmuş olmasıdır. Modern sentezin dayandığı genel bulgular 1930 ve 40'larda ortaya çıkan ve bugün kısmen DNA kopyalanması sırasındaki hatalarla oluştuğu bilinen mutasyon ve rekombinasyondur (Ebeveynlerine kıyasla farklı özellik kombinasyonları içeren bir yavru üretimi). Bunların dışında modern sentez, gen havuzunun genetik kayma ve gen akımı gibi mekanizmalarla değişime uğradığını ortaya koymuştur. Modern senteze göre, popülasyonlar çevresel nedenlerle (örneğin coğrafi engeller) birbirinden ayrıldığında türleşme meydana gelmektedir.

2.3 Genetik Algoritmanın Bileşenleri, Yapı ve Terminolojisi

Genetik algoritmalar biyolojik bir süreci simüle etmek için tasarlandığından, ilgili terminolojinin çoğu biyolojiden ödünç alınmıştır. Bununla birlikte, bu terminolojinin genetik algoritmalarda atıfta bulunduğu varlıklar, biyolojik karşılıklarından çok daha

basittir. Neredeyse tüm genetik algoritmalarda ortak olan temel bileşenler şunlardır (Carr 2014):

- optimizasyon için uygunluk fonksiyonu
- bir kromozom popülasyonu
- hangi kromozomların çoğalacağının seçimi
- yeni nesil kromozomlar üretmek için çaprazlama
- yeni nesil kromozomların rastgele mutasyonu

Uygunluk fonksiyonu, algoritmanın optimize etmeye çalıştığı fonksiyondur. "Uygunluk" kelimesi adını evrim teorisinden alınmıştır. Uygunluk fonksiyonu her bir potansiyel çözümün nasıl "uygun" olduğunu test eder ve ölçer. Uygunluk fonksiyonu, algoritmanın en önemli kısımlarından biridir ve algoritmada, kromozomların zamanla nasıl değişeceğini belirleyen tek adımdır. *Kromozom* terimi, genetik algoritmanın çözmeye çalıştığı probleme bir aday çözümü temsil eden sayısal bir değeri veya değerleri ifade eder. Her bir aday çözüm, diğer optimizasyon algoritmalarında da bulunan bir süreç olan, bir parametre değerler dizisi olarak kodlanır. Bir problem N boyutlu ise, tipik olarak her bir kromozom N elemanlı bir dizi olarak kodlanır.

$$kromozom = [p_1, p_2, \dots, p_N]$$

burada her p_i, i'nci parametrenin belirli bir değeridir. Aday çözümlerin örnek uzayının kromozomlara nasıl dönüştürüleceğini tasarlamak genetik algoritmanın kullanıcısına kalmıştır. Bir yaklaşım, her bir parametre değerini bir bit dizisine dönüştürmek (1'ler ve 0'lar dizisi), ardından kromozomları oluşturmak için bir DNA zincirindeki genler gibi uçtan uca parametreleri birleştirmektir. Tarihsel olarak, kromozomlar tipik olarak bu şekilde kodlanmıştır ve ayrık çözüm uzayları için uygun bir yöntem olmaya devam etmektedir. Modern bilgisayarlar, kromozomların permütasyonları, reel sayıları ve diğer birçok nesneyi içermesine izin verir. Problemin her olası çözümü bir bit dizisi olarak gösterilebiliyorsa, GA ile bu problemin çözülme ihtimali yüksektir. Her olası çözüme aday çözüm veya birey denir. Aday çözümlerden (veya kromozomlardan) oluşan bir diziye, GA'nın bir *popülasyonu* adı verilir. Genetik bir algoritma, ilk nesil (veya ilk popülasyon) olarak hizmet veren rastgele seçilmiş bir kromozom dizisi ile başlar. Daha

sonra popülasyondaki her bir kromozom (aday çözüm), problemi ne kadar iyi çözdüğünün belirlenmesi açısından, uygunluk fonksiyonu tarafından hesaplanır.

Seçim operatörü, kullanıcı tarafından tanımlanan bir olasılık dağılımına dayalı olarak üreme için bazı kromozomları seçer. Bir kromozom ne kadar uygunsa, seçilme olasılığı da o kadar yüksektir. Örneğin, f negatif olmayan bir uygunluk fonksiyonuysa, C_{53} kromozomunun yeniden üretilmek üzere seçilme olasılığı aşağıdaki gibi olabilir:

$$P(C_{53}) = \left| \frac{f(C_{53})}{\sum_{i=1}^{N} f(C_i)} \right|$$

Seçim operatörünün kromozomları değiştirme ile seçtiğini, bu nedenle aynı kromozomun birden fazla seçilebileceği unutulmamalıdır. *Çaprazlama operatörü*, hücre mayozunda kromozomların biyolojik geçişine ve rekombinasyonuna benzer. Bu operatör, iki yavru oluşturmak için seçilen kromozomlardan ikisinin bir alt dizisini değiştirir. Örneğin, ana kromozomlar

$$[11010111001000] \text{ ve } [01011101010010]$$

olsunlar. Yukarıdaki iki birey, tıpkı biyolojik popülasyonlardaki bireylerin çiftleşmesi gibi çiftleşebilir. Çiftleşmek için onların çaprazlaşmaları sağlanır, bu da her bireyin kendi genetik bilgilerinin bir kısmını yavrularıyla paylaştığı anlamına gelir. Çaprazlama noktasını bulmak için 1 ile 13 arasında rastgele bir sayı seçilir. Rastgele sayının 4 olduğunu varsayalım. Bu durumda dördüncü bitten sonraki çaprazlama sonucunda

$$[01010111001000] \text{ ve } [11011101010010]$$

yavruları (döller) elde edilir. Bu, iki bireyin her kromozomdaki dördüncü bit konumundan sonra tüm alellerini değiştirdiği anlamına gelir.

Mutasyon operatörü, yeni kromozomlardaki bireysel bitleri rastgele çevirir (0'ı 1'e, 1'leri 0'a). Tipik olarak mutasyon, 0.001 gibi çok düşük bir olasılıkla gerçekleşir. Bazı algoritmalar, mutasyon operatörünü seçim ve çaprazlama operatörlerinden önce uygular; bu bir tercih meselesidir. İlk bakışta, mutasyon operatörü gereksiz görünebilir. Aslında, seçim ve çaprazlama ikincil olsa bile önemli bir rol oynar. Seçim ve çaprazlama, daha uygun kromozomların genetik bilgisini korur, ancak bu kromozomlar

yalnızca mevcut nesle göre daha uygundur. Bu, algoritmanın çok hızlı yakınsamasına ve "potansiyel olarak yararlı genetik materyali" (belirli yerlerde 1'ler veya 0'lar), kaybetmesine neden olabilir (Goldberg D. E. 1989). Başka bir deyişle, algoritma global optimum bulmadan önce yerel bir optimumda takılıp kalabilir (Haupt 2004). Mutasyon operatörü, popülasyondaki çeşitliliği koruyarak bu soruna karşı korunmaya yardımcı olur, ancak aynı zamanda algoritmanın daha yavaş yakınsamasını da sağlayabilir.

Tipik olarak seçim, çaprazlama ve mutasyon süreci, yavru sayısı ilk popülasyonla aynı olana kadar devam eder, böylece ikinci nesil tamamen yeni yavrulardan oluşur ve ilk nesil tamamen değiştirilir. Bununla birlikte, bazı algoritmalar, birinci neslin oldukça uyumlu üyelerinin ikinci nesilde hayatta kalmasına izin verir.

Şimdi ikinci nesil uygunluk fonksiyonu ile test edilir ve döngü bu şekilde tekrar eder. Her nesilden en yüksek uyuma sahip kromozomu (uygunluk değeri ile birlikte) veya "şimdiye kadarki en iyi" kromozomu kaydetmek yaygın bir uygulamadır (Koza, 1994).

Genetik algoritmalar, "şimdiye kadarki en iyi" kromozomun uyum değeri sabitlenene ve birçok nesil boyunca değişmeyene kadar yinelenir. Bu, algoritmanın bir çözüm(ler)e yakınsadığı anlamına gelir. Tüm yineleme sürecine çalıştırma denir. Her çalışmanın sonunda, genellikle orijinal probleme oldukça uygun bir çözüm olan en az bir kromozom bulunur. Algoritmanın nasıl yazıldığına bağlı olarak, bu, tüm "şimdiye kadarki en iyi" kromozomların en uygun olanı veya son neslin en uygun olanı olabilir.
Yukarıdaki süreci daha iyi anlayabilmek için aşağıdaki problemi ele alalım:

Örnek 2. 1 Problemin, engebeli arazide gezinmek için yeterli güce ve ana üssüne çok sık dönmesi gerekmeyecek kadar yeterli süreye sahip (batarya anlamında), düşük ağırlıklı bir mobil robotun tasarımını içerdiğini varsayalım. Robot tasarımında belirtilmesi gereken parametreler arasında motor tipi ve boyutu ile güç kaynağı tipi ve boyutu yer almaktadır.
Motor tipi ve boyutu kodlaması:

000= 5V step motor,

001= 9V step motor,

010= 12V step motor,

011= 24V step motor

100= 5V servo motor,

101= 9V servo motor,

110= 12V servo motor,

111= 24V servo motor.

Güç kaynağı türü ve boyutları da aşağıdaki gibi kodlanabilir:

000=12V nikel-kadmiyum pil,

001=24V nikel-kadmiyum pil,

010=12V lityum iyon pil,

011=24V lityum iyon pil,

100=12V güneş paneli,

101=24V güneş paneli,

110=12V füzyon reaktörü,

111=24V füzyon reaktörü.

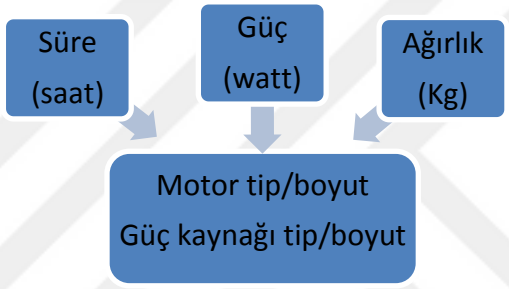
Sistem parametrelerinin kodlanması, GA'nın en önemli kısımlarından birisidir ve GA'nın gerçekten çalışıp çalışmadığı üzerinde önemli bir etkiye sahip olacaktır.

Bir kodlama şemasına karar verdikten sonra, her olası problem çözümünün "uygunluğunun" nasıl değerlendireceğine karar verilmelidir. Bu örnek için, robot ağırlığının, motor tipi/boyutu ve güç kaynağı tipi/boyutu ile ilişkili olduğu bir fonksiyon göz önüne alınabilir. Robot gücünü motorla ve güç kaynağıyla ilişkilendiren ve robotun çalışma süresini motorla ve güç kaynağıyla ilişkilendiren başka formüller de söz konusu olabilir. Eğer uygunluk için iyi bir tanımımız yoksa evrim simüle edilemez. Alternatif olarak, motor tipi/boyutu ve güç kaynağı tipi/boyutu girdileri için tasarımın ne kadar iyi

çalıştığının bir ölçüsünü çıkarabilecek bir bilgisayar simülasyonu mevcut olabilir. GA tasarımcısının problemi anlamasının kritik olduğu yer burasıdır. GA probleminin uygunluk fonksiyonunu tanımlamak için kesin kurallar yoktur. Mantıklı bir uygunluk fonksiyonu tanımlayabilmesi için problemi yeterince iyi anlamak GA tasarımcısına kalmıştır. Burada bir uygunluk fonksiyonu aşağıdaki gibi belirlenebilir:

$$\text{Uygunluk Fonksiyonu} = \text{Süre (saat)} + \text{Güç (Watt)} - \text{Ağırlık (kilogram)}$$

Çalışma süresi, güç ve ağırlık, motor tipi ve güç kaynağı tiplerine bağlı daha karmaşık fonksiyonlar göz önüne alınabilir veya uygunluk fonksiyonu, simülasyon yazılımı veya donanım deneylerinin çıktısı ile belirlenebilir (Bak. Şekil 2.2).



Şekil 2.2 Süre, güç ve ağırlığa bağlı motor tipi veya güç kaynağı.

GA' ya rastgele bir birey kümesi oluşturarak başlanır. Şimdi GA popülasyonundaki iki bireyi aşağıdaki gibi ele alalım. İlk birey 12 voltluk step motor ve 24 voltluk güneş panelli bir robot tasarımı, ikinci birey ise 9 voltluk servo motorlu ve 24 voltluk nikel-kadmiyum pilli bir robot tasarımıdır:

$$Birey\ 1 = \underbrace{12\ volt\ step\ motor}_{010},\ \underbrace{24\ volt\ güneş\ paneli}_{101}$$

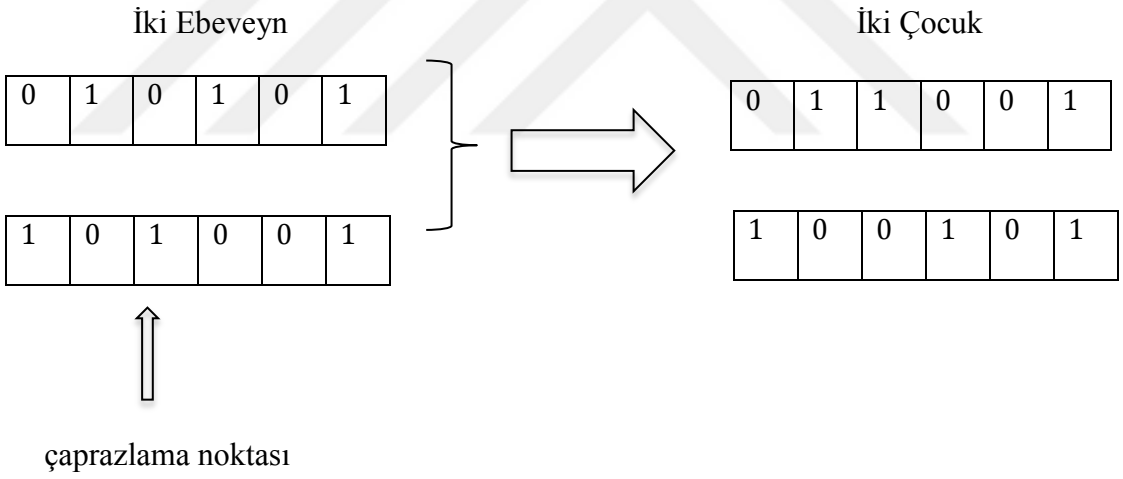
$$Birey\ 2 = \underbrace{9\ voltluk\ servo\ motor}_{101},\ \underbrace{24\ voltluk\ NiCad\ pil}_{001}.$$

Birey 1, bit dizisi 010101 ile kodlanır ve birey 2, bit dizisi 101001 ile kodlanır. Her bir bit, bir *alel* olarak adlandırılır. Bir bireyde, o bireyin bazı özellikleri hakkında bilgi içeren bit dizisi *gen* olarak adlandırılır. Spesifik genlere *genotip* denir ve bir genotipin temsil ettiği probleme özgü parametreye *fenotip* denir. Robot örneğimizde, biri motor boyutu/tipi için, diğeri pil boyutu/tipi için olmak üzere her bireyin iki geni vardır. Birey

1, "12 voltluk step motor" fenotipine karşılık gelen 010 motor genotipine ve "24 voltluk güneş pili" fenotipine karşılık gelen 101 pil genotipine sahiptir. Bir bireydeki tüm genlerin toplanmasıyla *kromozom* meydana gelir. Buna göre Birey 1, 010101 kromozomuna sahiptir.

Bir GA'da birçok birey olabilir. Tipik GA'ların düzinelerce veya yüzlerce bireyi vardır. Yukarıdaki iki birey, tıpkı biyolojik popülasyonlardaki bireylerin çaprazlanması gibi çaprazlanabilir. Çaprazlama sonunda her birey kendi genetik bilgilerinin bir kısmını yavrularıyla paylaşır. Çaprazlama noktasını bulmak için 1 ile 5 arasında rastgele bir sayı seçilir. Sayının 2 olduğunu kabul edelim (Birden fazla noktada da çaprazlama yapılabilir).

Bu, Şekil 2.3'de gösterildiği gibi, iki bireyin her kromozomdaki ikinci bit konumundan sonra tüm alellerini değiştireceği anlamına gelir.



Şekil 2.3 İkili bir GA'da çaprazlamanın gösterimi. Geçiş noktası rastgele seçilir. İki ebeveyn iki çocuk üretir.

İki ebeveyn, iki çocuk üretmek için çiftleşmiştir (yani çaprazlanmıştır). Her çocuk her iki ebeveynden de bazı genetik bilgiler alır. Ebeveynler ölür ve çocuklar evrim sürecini sürdürmek için hayatta kalır. Bu olaya GA'nın bir nesli denir.

Tıpkı biyolojide olduğu gibi, çocukların bazılarının uygunluğu yüksek, bazılarının uygunluğu düşük olacaktır. Düşük uygunluğu olan bireylerin kendi nesillerinde ölme

olasılığı yüksektir; yani, GA simülasyonundan çıkarılırlar. Yüksek uygunluklu bireyler, diğer yüksek uygunluklu bireylerle çaprazlamak için hayatta kalır ve böylece yeni nesil bireyler üretir. GA, optimizasyon problemine kabul edilebilir bir çözüm bulana kadar bu işleme devam edilir. GA yazılımında bir noktada, hangi bireylerin çocuk üretmek için eşleştirileceğine karar verilmesi gereklidir. Bu karar, popülasyondaki bireylerin uygunluğuna dayanmaktadır. En uygun bireylerin çiftleşerek çocuk üretmesi olasıyken, uygun olmayan bireylerin eş bulması pek olası değildir ve bu nedenle herhangi bir yavru üretemeden ölmeleri muhtemeldir.

Ebeveynleri seçmenin yaygın bir yolu, uygunluk-orantılı seçim olarak da adlandırılan rulet tekerleği seçimidir. Popülasyonumuzda dört birey olduğunu varsayalım (Gerçek bir GA'da dörtten fazla birey olacaktır). Bireysel uygunlukların aşağıdaki gibi değerlendirildiğini varsayalım:

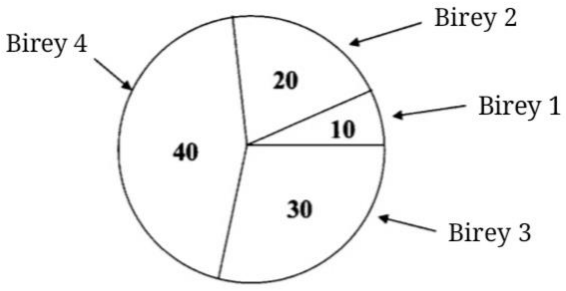
Birey 1: Uygunluk = 10

Birey 2: Uygunluk = 20

Birey 3: Uygunluk = 30

Birey 4: Uygunluk = 40

4. birey en uygun ve 1. birey en az uygun olandır. Her bir slot alanı bireylerden birinin uygunluğuna karşılık gelen bir rulet çarkı oluşturur. Örneğimizde rulet çarkı aşağıdaki şekilde gösterilmiştir.



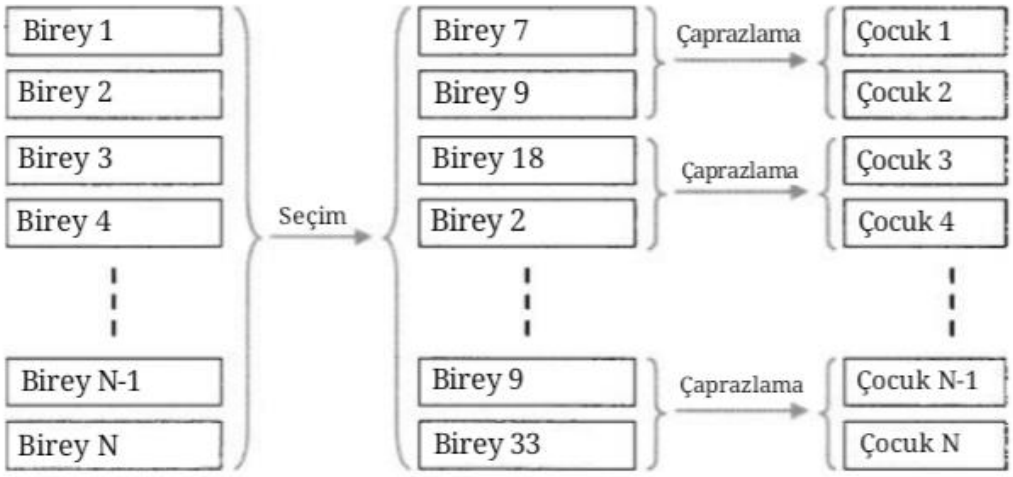
Şekil 2.4 Yukarıdaki pasta grafik, dört üyeli bir popülasyon için bir GA' da rulet tekerleği seçimini göstermektedir. Her bireye uygunluğuyla orantılı bir dilim atanır. Her bireyin ebeveyn olarak seçimi, rulet çarkındaki dilimiyle orantılıdır

Bazen bir uygunluk fonksiyonu, tüm bireylerin birbirine çok yakın uygunluk değerlerine sahip olacak şekilde zayıf bir şekilde tanımlanır. Uygunluk değerleri bir araya toplanırsa, seçim süreci yüksek ve düşük uyumlu bireyler arasında iyi bir ayrım yapamaz. Bu, daha uygun bireylerin bir sonraki nesle yayılmasını engeller. Bazen tersine bir sorun da şu şekilde ortaya çıkar; uygunluk değerleri çok fazla dağılmıştır, bu nedenle düşük uygunluktaki bireylerin üreme için seçilme şansları yoktur.

Popülasyon boyutunu bir nesilden diğerine sabit tutmak için iki çift eş seçilir. Bunlar toplam dört çocuk üretecektir. İlk eş çiftini seçmek için, hayali (bilgisayar simülasyonlu) bir çark döndürülür ve rulet çarkında nereye giderse gitsin, ilk ebeveynin kim olduğu böylece belirlenir. Şekil 2.4'deki rulet çarkından, 1. bireyin %10 seçilme şansına, 2. bireyin %20 seçilme şansına, 3. bireyin de %30'luk bir seçilme şansına sahip olduğunu, başka bir deyişle her bireyin uygunluğu ile orantılı seçilme olasılığına sahip olduğunu görülmektedir. Ardından, ikinci bir ebeveyn seçmek için rulet çarkı ikinci kez çevrilir. Rulet çarkı ilk dönüşle aynı kişide durursa, çark tekrar döndürülür (bir ebeveyn kendisiyle çiftleşemez). İki ebeveyn belirlendikten sonra, iki çocuk üretmek üzere çaprazlanırlar. Daha sonra iki ebeveyn daha elde etmek, onları çiftleştirmek ve iki çocuk daha elde etmek için işlem tekrarlanır. Bu süreç, çocuk nüfusu, ebeveyn nüfusu ile aynı büyüklüğe gelene kadar tekrar etmeye devam eder. Bu durum Şekil 2.5'te gösterilmektedir.

Şekil 2.4'de gösterilen uygunluk değerleri göz önüne alındığında, bir ebeveyn seçmek için rulet çarkını döndürme işlemi Şekil 2.5'te gösterildiği gibi gerçekleştirilebilir. Gelecek nesil için dört çocuk oluşturan dört ebeveyn seçmek için Şekil 2.5'teki işlem dört kez tekrarlanır.

Şekil 2.6, N bireylik bir popülasyon verildiğinde, rulet tekerleği seçimini kullanarak bir ebeveyni seçen yapay kodu gösterir. Gelecek nesillerde çocukların yaratılması için ebeveynleri seçmek için Şekil 2.6'daki süreç gerektiği kadar tekrarlanır.



Şekil 2.5 Bir çocuk popülasyonu oluşturmak için ebeveyn popülasyonunun çaprazlanmasının gösterimi

Soldaki N bireyden oluşan ilk popülasyon, bir N ebeveyn kümesi oluşturmak için bir seçim sürecinden (örneğin, rulet tekerleği seçiminden) geçer. Bazı bireyler birden fazla seçilebilirken, bazıları hiç seçilmeyebilir. Sonra ortadaki her bir ebeveyn çifti bir çift çocuk yaratmak için çaprazlanır

```
Generate a uniformly distributed random number r ∈ [0, 1]
If r < 0.1 then
    Parent = Individual 1
else if r < 0.3 then
    Parent = Individual 2
else if r < 0.6 then
    Parent = Individual 3
else
    Parent = Individual 4
End if
```

Şekil 2.6 Şekil 2.4'deki rulet çarkına göre bir ebeveynin seçiliminin yapay kodu.

```
x_i = i-th individual in population, i ∈ [1, N]
f_i ← fitness(x_i) for i ∈ [1, N]
f_sum = Σ_{i=1}^{N} f_i
Generate a uniformly distributed random number r ∈ [0, f_sum]
F ← f_1
k ← 1
While F < r
    k ← k + 1
    F ← F + f_k
End while
Parent ← x_k
```

Şekil 2.7 Rulet tekerleği seçimi kullanılarak N bireyden bir ebeveynin nasıl seçileceğini gösteren yapay kod. Bu kodda, $\forall i \in (1, N]$ için uygunluk değerinin $f_i \geq 0$ olduğunu varsayılmıştır

GA uygulamalarında bazen araştırmacılar, yalnızca birbirlerine yeterince benzeyen bireylerin, yani, yalnızca aynı türe ait bireylerin çiftleşmesine izin verirken, bazen de bireylerin yalnızca birbirlerinden yeterince farklı olmaları durumunda yani, yalnızca farklı "ailelere" aitlerse, çiftleşmelerine izin vermektedirler. Diğer taraftan burada ele alınan rulet tekerleği seçiminin dışında, turnuva seçimi, sıra seçimi ve diğer birçok varyasyon dahil olmak üzere başka seçim türleri de mevcuttur.

GA'daki son adım mutasyondur. Biyolojide mutasyon, en azından yavruları gözle görülür şekilde etkilediği sürece, nispeten nadirdir. Çoğu GA uygulamasında da mutasyon nadirdir (%2 derecesinde). Genel olarak bir GA mutasyon oranı için doğru ayarın ne olması gerektiği hakkında net bir bilgi yoktur. En iyi mutasyon oranı probleme, popülasyon büyüklüğüne, kodlamaya ve diğer faktörlere bağlıdır. Frekansı ne olursa olsun, mutasyon önemlidir çünkü evrimsel sürecin probleme yeni potansiyel çözümler keşfetmesine izin verir. Eğer popülasyonda bazı genetik bilgiler eksikse, mutasyon bu bilgiyi popülasyona dahil etme imkanını sağlar. Bu biyolojik evrimde önemlidir, ancak GA'larda daha da önemlidir. Bunun nedeni, GA'ların tipik olarak o kadar küçük popülasyon boyutlarına sahip olmalarıdır ki, aynı soydan çiftleşme (akraba üremesi) kolayca bir problem haline gelebilir ve evrimsel çıkmaz sokaklar GA'larda biyolojik evrimden daha yaygındır. Biyolojik evrimde tipik olarak milyonlarca popülasyondan söz konusuyken, GA'larda düzinelerce veya yüzlerce popülasyon söz konusudur.

Mutasyonu uygulamak için bir mutasyon olasılığı seçilir. Örneğin %1 lik bir olasılık, çaprazlama işlemi yavrular ürettikten sonra, her çocuktaki her bitin %1'lik bir ters değere çevrilme olasılığına sahip olduğu anlamına gelir (1'ler, 0'a veya 0'lar, 1'le değişir). Mutasyon basittir, ancak makul bir mutasyon olasılığı seçmek önemlidir. Mutasyon olasılığının çok yüksek olması, GA'nın rastgele bir arama gibi davranmasına neden olur ve bu bir problemin çözümünde genellikle iyi bir yol değildir. Mutasyon olasılığının çok düşük olması, akrabalı yetiştirme ve evrimsel çıkmazlarla ilgili sorunlara neden olur ve bu da GA'nın iyi bir çözüm bulmasını engeller.

Her bireyin n bitinin olduğu ve ρ mutasyon oranlı, N bireyli bir x_i popülasyonunu göz önüne alalım. Buna göre her neslin sonunda, her bir bireydeki her bit ρ olasılığı ile aşağıdaki gibi dönüştürülebilir:

$$r \leftarrow U[0,1]$$

$$x_i(k) \leftarrow \begin{cases} x_i(k) & \text{eğer } r \geq \rho \\ 0 & \text{eğer } r < \rho \text{ ve } x_i(k) = 1 \\ 1 & \text{eğer } r < \rho \text{ ve } x_i(k) = 0 \end{cases}$$

Burada $i \in [1, N]$, $k \in [1, n]$ için, $U[0,1]$, $[0,1]$ üzerinde düzgün dağılımlı rastgele bir sayıdır.

Şekil 2.8 basit bir GA'yı özetlemektedir. Bu esnek algoritma üzerinde birçok değişiklik yapılabilir. Örneğin, bir GA için durdurma kriterleri birkaç farklı seçenek. Bir olasılık, GA'nın önceden belirlenmiş sayıda nesil boyunca çalışabilmesidir. Bir başka olasılık da, GA'nın en iyi bireyin uygunluğu, kullanıcı tanımlı bir eşikten daha iyi olana kadar çalışmasıdır. Eğer problemimiz "yeterince iyi" bir çözüm bulmaksa, bu makul bir durdurma kriteri olabilir. Bir başka olasılık, GA'nın en iyi bireyin uygunluğu bir nesilden diğerine gelişmeyi durdurana kadar çalışmasıdır. Bu, evrim sürecinin bir platoya ulaştığını ve daha fazla gelişemediğini gösterir.

```
Parents ← {randomly generated population}
While not(termination criterion)
    Calculate the fitness of each parent in the population
    Children ← ∅
    While |Children| < |Parents|
        Use fitnesses to probabilistically select a pair of parents for mating
        Mate the parents to create children c1 and c2
        Children ← Children ∪ {c1, c2}
    Loop
    Randomly mutate some of the children
    Parents ← Children
Next generation
```

Şekil 2.8 Basit bir GA algoritması.

3. GENETİK ALGORİTMA UYGULAMALARI

Bu bölümde önceki bölümde teorisi ayrıntılı bir şekilde verilen Genetik Algoritmaların farklı problemlere uygulamalarına değinilecektir.

Örnek 3.1 Sayısal bir örnek (Hermawanto, 2013)

$x + 2y + 3z + 4t = 30$ denklemini göz önüne alalım. Bu denklemi sağlayan x, y, z ve t değerini bulmak için genetik algoritma kullanılacaktır. Öncelikle amaç fonksiyonu formülize edilmelidir. Bu problem için $f(x, y, z, t) = |(x + 2y + 3z + 4t) - 30|$ olarak alınırsa, amaç $f(x, y, z, t)$ fonksiyonunun değerini en aza indirecek x, y, z ve t parametrelerinin belirlenmesidir. Bilinmeyen dört değişken olduğundan, kromozom

[x; y; z; t]

şeklinde oluşturulabilir. Hesaplamayı hızlandırmak için x, y, z ve t değişkenlerinin değerleri 0 ile 30 arasında tamsayılar olacak şekilde kısıtlanacaktır.

Adım 1. Başlangıç Değerleri

Popülasyondaki kromozom sayısının 6 olduğunu kabul edelim. Buna göre 6 kromozom için x, y, z, t geninin rastgele değerleri aşağıdaki gibi oluşturulabilir:

Kromozom [1] = [x; y; z; t] = [12; 05; 23; 08]

Kromozom [2] = [x; y; z; t] = [02; 21; 18; 03]

Kromozom [3] = [x; y; z; t] = [10; 04; 13; 14]

Kromozom [4] = [x; y; z; t] = [20; 01; 10; 06]

Kromozom [5] = [x; y; z; t] = [01; 04; 13; 19]

Kromozom [6] = [x; y; z; t] = [20; 05; 17; 01]

Adım 2. Hesaplama

Bu kısımda başlangıç adımında üretilen her kromozom için amaç fonksiyon değerleri hesaplanır. Sonuçlar Çizelge 3.1'de verilmiştir.

Çizelge 3.1 Amaç Fonksiyon Değer Çizelgesi

Kromozomlar	Genler (Değerler)				Amaç Fonksiyonu	
	x	y	z	t	$(x + 2\,y + 3\,z + 4\,t) - 30$	
1	12	05	23	08	F_obj[1]	93
2	02	21	18	03	F_obj[2]	80
3	10	04	13	14	F_obj[3]	83
4	20	01	10	06	F_obj[4]	46
5	01	04	13	19	F_obj[5]	94
6	20	05	17	01	F_obj[6]	55

Adım 3. Seçilim

Bu aşama, mevcut popülasyondan sonraki aşamalara geçerek yeni popülasyonu oluşturacak en uygun kromozomların seçilimi aşamasıdır. En uygun kromozomlar gelecek nesil için daha yüksek seçilme olasılığına sahiptir. Uygunluk olasılığını hesaplamak için her bir kromozomun uygunluğu hesaplanmalıdır. Sıfır ile bölünme probleminden kaçınmak için, F_obj' nin değerine 1 eklenir.

$$Fitness\,[i] = 1/\left(1 + F_{obj}[i]\right), \qquad i = 1, \ldots, 6$$

$$Toplam = \sum_{i=1}^{6} Fitness\,[i]$$

Her bir kromozomun olasılığı

$$P\,[i] = Fitness\,[i] \,/\, Toplam$$

formülü yardımıyla hesaplanırsa bulunan değerler aşağıdaki Çizelge 3.2'deki gibi elde edilir. Çizelge 3.2'deki olasılıklardan, en yüksek uygunluğa sahip olan Kromozom 4'ün,

yeni nesil kromozomlar için en yüksek seçilme olasılığına sahip olduğu görülebilmektedir.

Seçim süreci için en bilinen yöntemlerden biri rulet çarkı yöntemidir. Rulet çarkı seçiliminde, topluluktaki tüm bireylerin uygunluk değerleri toplanır ve her bireyin seçilme olasılığı, uygunluk değerinin bu toplam değere oranı kadardır. Bunun için birikimli olasılık değerleri hesaplandıktan sonra Çizelge 3.3 elde edilir.

Çizelge 3.2 Uygunluk Olasılığı Değer Çizelgesi

Amaç Fonksiyonu	Değerler	Uygunluk Olasılığı	(1/ (1+F_obj))
F_obj[1]	93	Fitness[1]	0.010638298
F_obj[2]	80	Fitness[2]	0.012345679
F_obj[3]	83	Fitness[3]	0.011904762
F_obj[4]	46	Fitness[4]	0.021276596
F_obj[5]	94	Fitness[5]	0.010526316
F_obj[6]	55	Fitness[6]	0.017857143

Çizelge 3.3 Birikimli Olasılık Değer Çizelgesi

Amaç Fonksiyonu	Bireysel Olasılığı (P[i] =Fitness[i] / Total)		Birikimli Olasılığı	
F_obj[1]	P[1]	0.125824361	C[1]	0.125824
F_obj[2]	P[2]	0.146018394	C[2]	0.271843
F_obj[3]	P[3]	0.140803452	C[3]	0.412646
F_obj[4]	P[4]	0.251648722	C[4]	0.664295
F_obj[5]	P[5]	0.124499894	C[5]	0.788795
F_obj[6]	P[6]	0.211205177	C[6]	1

Rulet çarkına göre ilk olarak $[0,1]$ aralığında rastgele $R[i]$ sayıları aşağıdaki gibi üretilir:

R [1] = 0.201, R [2] = 0,284, R [3] = 0,099, R [4] = 0,822, R [5] = 0,398, R [6] = 0.501

Eğer rastgele sayı R[1], C[1] 'den büyük ve C[2]' den küçükse, sonraki nesil için yeni popülasyonda kromozom olarak Kromozom [2] seçilir, yani;

YeniKromozom [1] = Kromozom[2] (C[1]<R[1]<C[2] olduğundan)

olur ve benzer olarak diğer yeni kromozomlar

YeniKromozom [2] = Kromozom[3] (C[2]<R[2]<C[3] olduğundan)

YeniKromozom [3] = Kromozom[1] (R[3]<C[1] olduğundan)

YeniKromozom [4] = Kromozom[6] (C[5]<R[4]<C[6] olduğundan)

YeniKromozom [5] = Kromozom[3] (C[2]<R[5]<C[3] olduğundan)

YeniKromozom [6] = Kromozom[4] (C[3]<R[6]<C[4] olduğundan)

şeklinde seçilir. Böylece popülasyondaki kromozomlar aşağıdaki şekilde yeniden düzenlenmiş olurlar:

Kromozom[1] = [02; 21; 18; 03]

Kromozom[2] = [10; 04; 13; 14]

Kromozom[3] = [12; 05; 23; 08]

Kromozom[4] = [20; 05; 17; 01]

Kromozom[5] = [10; 04; 13; 14]

Kromozom[6] = [20; 01; 10; 06]

Adım 4. Çaprazlama

Bu örnekte, tek noktalı çaprazlama kullanılacaktır, yani ana kromozomda rastgele bir konum seçilip alt kromozomlar değiştirilecektir. Eşlenecek ebeveyn kromozom rastgele

seçilir ve çoğalma kromozomlarının sayısı çaprazlama_oranı (ρc) parametreleri kullanılarak kontrol edilir. Çaprazlama işlemi için algoritma aşağıdaki gibidir:

Begin
k← 0;
while(k<population) do
R[k] ← random(0-1);
if (R[k] < ρc) then
select Chromosome[k] as parent;
end;
k = k + 1;
end;
end;

$R[k] < \rho c$ ise, $kromozom[k]$ ebeveyn olarak seçilecektir. Çaprazlama oranı % 25 olarak belirlensin. Buna göre eğer $kromozom[k]$ için rasgele oluşturulan değer 0.25'in altındaysa çaprazlama için k kromozom numarası seçilecektir. Süreç aşağıdaki gibidir:

İlk olarak nüfus sayısı kadar rastgele bir R sayısı oluşturulur.

$$R\,[1] = 0{,}191, R\,[2] = 0{,}259, R\,[3] = 0{,}760, R\,[4] = 0{,}006,$$

$$R\,[5] = 0{,}159, R\,[6] = 0{,}340$$

Yukarıdaki rastgele R sayısı için ebeveynler $Kromozom[1]$, $Kromozom[4]$ ve $Kromozom[5]$ çaprazlama için seçilecektir.

$$Kromozom\,[1] >< Kromozom\,[4]$$

$$Kromozom\,[4] >< Kromozom\,[5]$$

$$Kromozom\,[5] >< Kromozom\,[1]$$

Kromozom seçiminden sonra, bir sonraki süreç çaprazlama noktasının konumunu belirlemektir. Bu, 1 ile (Kromozom uzunluğu - 1) arasında rastgele sayılar oluşturularak yapılır. Bu durumda, üretilen rastgele sayılar 1 ile 3 arasında olmalıdır. Çaprazlama noktasını aldıktan sonra, ebeveyn kromozomu çaprazlama noktasında kesilecek ve genleri değiştirilecektir. Örneğin rastgele 3 sayı aşağıdaki gibi üretilir;

$$C\,[1]\ =\ 1, C\,[2]\ =\ 1, C\,[3]\ =\ 2$$

Daha sonra, birinci çaprazlama, ikinci çaprazlama ve üçüncü çaprazlama için, ebeveynin genleri sırasıyla 1.gen, 1. gen ve 2.gen numarasından kesilecektir;

$$Kromozom[1]\ =\ Kromozom[1] >< Kromozom[4]$$

$$=\ [02;\ 21;\ 18;\ 03] >< [20;\ 05;\ 17;\ 01]$$

$$=\ [02;\ 05;\ 17;\ 01]$$

$$Kromozom[4]\ =\ Kromozom[4] >< Kromozom[5]$$

$$=\ [20;\ 05;\ 17;\ 01] >< [10;\ 04;\ 13;\ 14]$$

$$=\ [20;\ 04;\ 13;\ 14]$$

$$Kromozom[5]\ =\ Kromozom[5] >< Kromozom[1]$$

$$=\ [10;\ 04;\ 13;\ 14] >< [02;\ 21;\ 18;\ 03]$$

$$=\ [10;\ 04;\ 18;\ 03]$$

Böylece, bir çaprazlama süreci sonunda Kromozom popülasyonu aşağıdaki gibi elde edilir:

$$Kromozom[1]\ =\ [02;\ 05;\ 17;\ 01]$$

$$Kromozom[2]\ =\ [10;\ 04;\ 13;\ 14]$$

$$Kromozom[3]\ =\ [12;\ 05;\ 23;\ 08]$$

$$Kromozom[4]\ =\ [20;\ 04;\ 13;\ 14]$$

$$Kromozom[5]\ =\ [10;\ 04;\ 18;\ 03]$$

$$Kromozom[6]\ =\ [20;\ 01;\ 10;\ 06]$$

Adım 5. Mutasyon

Bir popülasyonda mutasyona sahip olan kromozomların sayısı, mutasyon oranı parametresi ile belirlenir. Mutasyon işlemi, rastgele pozisyondaki genin yeni bir değer ile değiştirilmesiyle yapılır. Süreç aşağıdaki gibidir:

İlk önce popülasyondaki toplam gen uzunluğunu hesaplanmalıdır. Bu durumda genin toplam uzunluğu;

$$toplam\_gen = kromozomdaki\ gen\ sayısı * popülasyon\ sayısıdır$$

$$= 4 * 6 = 24$$

Mutasyon işlemi, 1 ile toplam gen (1'den 24'e) arasında rastgele bir tamsayı oluşturarak yapılır. Oluşturulan rastgele sayı mutasyon oranı (ρm) değişkeninden küçükse, o zaman genin kromozomlardaki konumu işaretlenir. Pm'i %10 tanımladığımızı varsayalım. Bu durumda mutasyona uğrayacak popülasyondaki toplam_gen'in % 10 (0.1) olması beklenir:

$$mutasyon\ sayısı = 0.1 * 2.4 = 2.4 \approx 2$$

Şimdi rastgele sayı üretiminin 12 ve 18 ürettiğini kabul edelim. Bu durumda mutasyona sahip kromozomların sırasıyla Kromozom numarası 3 gen numarası 4 ve Kromozom numarası 5 gen numarası 2 dir. Mutasyon noktasındaki mutasyona uğramış genlerin değeri 0-30 arasındaki rastgele sayılar ile değiştirilir.

Oluşturulan rastgele sayıların 2 ve 5 olduğunu varsayarsak, mutasyondan sonraki Kromozom popülasyonu:

$$Kromozom[1] = [02; 05; 17; 01]$$

$$Kromozom[2] = [10; 04; 13; 14]$$

$$Kromozom[3] = [12; 05; 23; 02]$$

$$Kromozom[4] = [20; 04; 13; 14]$$

$$Kromozom[5] = [10; 05; 18; 03]$$

$$Kromozom[6] = [20; 01; 10; 06]$$

olarak elde edilir.

İterasyon

Mutasyon işlemi bittiğinde, artık bir yinelemeye veya bir genetik algoritma nesline sahip olunmuştur. Amaç fonksiyonu bu yeni nesil için yeniden hesaplanırsa:

$$F_{obj}\,[1] \;=\; Abs((02 + 2 * 05 + 3 * 17 + 4 * 01) - 30) = 37$$

$$F_{obj}\,[2] \;=\; Abs((10 + 2 * 04 + 3 * 13 + 4 * 14) - 30) = 77$$

$$F_{obj}\,[3] \;=\; Abs((12 + 2 * 05 + 3 * 23 + 4 * 02) - 30) = 47$$

$$F_{obj}[4] \;=\; Abs\big((20 + 2 * 04 + 3 * 13 + 4 * 14) - 30\big) = 93$$

$$F_{obj}[5] \;=\; Abs\big((10 + 2 * 05 + 3 * 18 + 4 * 03) - 30\big) = 56$$

$$F_{obj}\,[6] = Abs\big((20 + 2 * 01 + 3 * 10 + 4 * 06) - 30\big) = 46$$

elde edilir.

Yeni kromozom incelenirse amaç fonksiyonunun değerlerindeki azalma görülebilmektedir. Bu da önceki kromozom nesline kıyasla daha iyi kromozom veya çözüme sahip olunduğu anlamına gelir. Bir sonraki yineleme için yeni kromozomlar aşağıdaki gibidir:

$$Kromozom\,[1] \;=\; [02;\ 05;\ 17;\ 01]$$

$$Kromozom\,[2] \;=\; [10;\ 04;\ 13;\ 14]$$

$$Kromozom\,[3] \;=\; [12;\ 05;\ 23;\ 02]$$

$$Kromozom\,[4] \;=\; [20;\ 04;\ 13;\ 14]$$

$$Kromozom\,[5] \;=\; [10;\ 05;\ 18;\ 03]$$

$$Kromozom\,[6] \;=\; [20;\ 01;\ 10;\ 06]$$

Bu yeni kromozomlar, değerlendirme, seçim, çaprazlama ve mutasyon gibi önceki nesil kromozomlarla aynı süreçten geçecek ve sonunda gelecek yineleme için yeni nesil kromozom üretecek ve bu süreç, önceden belirlenen sayıda nesil sayısı kadar tekrarlanacaktır. Bu örnek için 50 nesil çalıştıktan sonra en iyi kromozom aşağıdaki şekilde elde edilir:

$$Kromozom \;=\; [07;\ 05;\ 03;\ 01]$$

Böylece, $x = 7, y = 5, z = 3, t = 1$ olarak elde edilir. Gerçekten;

$$x + 2y + 3z + 4t = 7 + (2 * 5) + (3 * 3) + (4 * 1) = 30$$

olduğu görülebilir.

Örnek 3.2 Tek değişkenli bir fonksiyonun optimizasyonu (Michalewicz 1992)

Bu örnekte, tek değişkenli

$$f(x) = xSin(10\pi x) + 1.0$$

fonksiyonunun optimizasyonu için bir genetik algoritma yaklaşımı ele alınacaktır. Problem, $[-1,2]$ aralığında f fonksiyonunu maksimize eden x'i bulmaktır. f fonksiyonunun analizi nispeten kolaydır. f' birinci türevinin sıfırları belirlenirse

$$\tan(10\pi x) = -10\pi x$$

eşitliği elde edilir. Yukarıdaki denklemin sonsuz sayıda çözüme sahip olduğu açıktır;

$$x_i = \frac{2i-1}{20} + \epsilon_i, \quad i = 1, 2, \ldots$$

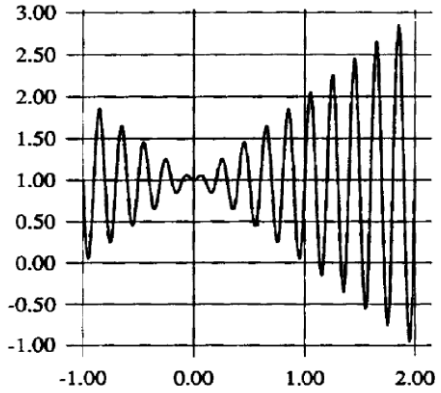
$$x_0 = 0$$

$$x_i = \frac{2i+1}{20} - \epsilon_i, \quad i = -1, -2, \ldots$$

Burada ϵ_i terimleri, sıfıra yaklaşan azalan reel sayı dizilerini ($i = 1, 2, \ldots$ ve $i = -1, -2, \ldots$için) temsil eder.

Eğer i bir tek tamsayı ise x_i, f için bir yerel maksimum ve eğer i bir çift tamsayı ise x_i, f için bir yerel minimum olacaktır (bkz. Şekil 3.1).

Problemin tanım kümesi $x \in [-1, 2]$ olduğundan, fonksiyon $x_{19} = \frac{37}{20} + \epsilon_{19} = 1.85 + \epsilon_{19}$ için maksimuma ulaşır. Burada $f(x_{19}), f(1.85) = 1.85 \sin\left(18\pi + \frac{\pi}{2}\right) + 1.0 =$ 2.85'den biraz daha büyük bir değerdir.



Şekil 3.1 $f(x) = xSin(10\pi x) + 1.0$ fonsiyonunun grafiği

Şimdi yukarıdaki problemi çözmek için, yani f fonksiyonunu maksimize etmek için bir genetik algoritma oluşturmak istediğimizi varsayalım.

Adım 1. Temsil

x reel değerlerinin temsilinde kromozom olarak ikili vektör gösterimi kullanılacaktır. Vektörün uzunluğu, bu örnek için, ondalık noktadan altı basamak sonra olan gerekli hassasiyete bağlıdır. x değişkeninin tanım kümesi 3 birim uzunluğuna sahiptir; kesinlik gereksinimi [-1,2] aralığının en az $3 * 1000000$ eşit boyut aralığına bölünmesi gerektiğini belirtir. Bu, ikili vektör (kromozom) olarak 22 bit gerektiği anlamına gelir:

$$2097152 = 2^{21} < 3000000 < 2^{22} = 4194304$$

İkili bir $\langle b_{21} b_{20} \ldots b_0 \rangle$ dizisinden $[-1, 2]$ aralığındaki bir x reel sayısına dönüşüm yapmak mümkündür ve bu işlem iki adımda tamamlanır:

• ikili dizi $\langle b_{21} b_{20} \ldots b_0 \rangle$ 2 tabanından 10 tabanına dönüştürülür:

$$\left(\langle b_{21} b_{20} \ldots bo \rangle\right)_2 = \left(\sum_{i=0}^{21} b_i \cdot 2^i \right)_{10} = x',$$

• karşılık gelen x reel sayısı bulunur:

$$x = -1,0 + x' \cdot \frac{3}{2^{22} - 1}$$

Örneğin, kromozom

$$(1000101110110101000111),$$

0.637197 sayısını temsil eder;

$$x' = (1000101110110101000111)_2 = 2288967$$

ve

$$x = -1,0 \; + \; 2288967 \cdot \frac{3}{4194303} = \; 0,637197.$$

(00000000000000000000) ve (11111111111111111111) kromozomları da sırasıyla 1.0 ve 2.0 aralık sınırlarını temsil eder.

Başlangıç popülasyonu

Başlangıç süreci basittir: Her bir kromozomun 22 bitlik bir ikili vektör olduğu bir kromozom popülasyonu oluşturulur. Her bir kromozomun 22 biti, rastgele seçilerek başlanır.

Değerlendirme fonksiyonu

v ikili vektörü için $eval$ değerlendirme fonksiyonu, f fonksiyonuna eşittir;

$$eval(v) \; = f(x),$$

burada v kromozomu, x reel değerini temsil eder.

Daha önce belirtildiği gibi, değerlendirme fonksiyonu, potansiyel çözümleri uygunlukları açısından derecelendiren çevrenin rolünü oynar. Örneğin, aşağıdaki üç kromozom

$$\begin{aligned} v_1 &= (1000101110110101000111), \\ v_2 &= (0000001110000000010000), \\ v_3 &= (1110000000111111000101), \end{aligned}$$

sırasıyla $x_1 = 0.637197$, $x_2 = -0.958973$ ve $x_3 = 1.627888$ değerlerine karşılık gelir. Sonuç olarak, değerlendirme fonksiyonu bunları şu şekilde derecelendirir:

$$eval(v_1) = f(x_1) = 1.586345,$$
$$eval(v_2) = f(x_2) = 0,078878,$$
$$eval(v_3) = f(x_3) = 2.250650.$$

Açıkça, v_3 kromozomu, en yüksek değeri verdiği için üç kromozomdan en iyisidir.

Genetik operatörler

Genetik algoritmanın değiştirme aşamasında, klasik iki genetik operatör; mutasyon ve çaprazlama kullanılacaktır.

Daha önce belirtildiği gibi, mutasyon, bir veya daha fazla geni (bir kromozomdaki pozisyonları), mutasyon oranına eşit bir olasılıkla değiştirir. Bir mutasyon için v_3 kromozomundan beşinci genin seçildiğini varsayalım. Bu kromozomdaki beşinci gen 0 olduğundan, 1'e çevrilir. Yani bu mutasyondan sonraki v_3 kromozomu şöyle gösterilir:

$$v_3' = (11101000001111111000101).$$

Bu kromozom, $x_3' = 1.721638$ ve $f(x_3') = -0.082257$ değerini temsil eder. Bu mutasyonun, kromozom v_3'ün değerinde önemli bir düşüşe neden olduğu açıkça görülmektedir. Öte yandan, v_3 kromozomundaki mutasyon için 10. geni seçilmişse,

$$v_3'' = (11100000011111111000101).$$

olup, karşılık gelen değer $x_3'' = 1.630818$ ve $f(x_3'') = 2.343555$ olarak elde edilir. Bu durumda $f(x_3) = 2.250650$'nin orijinal değerine göre bir gelişme kaydedilmiştir.

v_2 ve v_3 kromozomları üzerindeki çaprazlama operatörünü ele alalım. Çaprazlama noktasının 5. genden sonra (rastgele) seçildiğini varsayalım:

$$v_2 = (00000|01110000000010000),$$
$$v_3 = (11100|00000111111000101).$$

Ortaya çıkan iki yavru

$$v_2' = (00000|00000111111000101),$$
$$v_3' = (11100|01110000000010000).$$

dür. Bu yavruların değerleri aşağıdaki gibi elde edilir;

$$f(v_2') = f(-0,998113) = 0,940865,$$
$$f(v_3') = f(1.666028) = 2.459245.$$

İkinci yavrunun her iki ebeveyninden daha iyi bir değerlendirmeye sahip olduğuna dikkat edelim.

Parametreler

Bu problem için parametreler şu şekilde kullanılmıştır: $pop\_size = 50$, çaprazlama olasılığı $P_c = 0.25$, ve mutasyon olasılığı $P_m = 0.01$. Aşağıda, böyle bir genetik sistem için bazı deneysel sonuçları sunulmaktadır:

Deneysel sonuçlar

Çizelge 3.4'de, değerlendirme fonksiyonunda hesaplanan nesil sayıları, karşılık gelen fonksiyonun değeriyle birlikte sunulmaktadır. 150 nesil sonra en iyi kromozom

$$v_{max} = (11110011010001000001 01)$$

olup, reel karşılığı $x_{max} = 1.850773$ olarak elde edilir. Beklendiği gibi, $x_{max} = 1.85 + \epsilon$, ve $f(x_{max})$, 2.85'ten biraz daha büyüktür.

Çizelge 3.4 150 nesil sonuçları

Nesil sayısı	Değerlendirme fonksiyonu
1	1.441942
6	2.250003
8	2.250283
9	2.250284
10	2.250363
12	2.328077
39	2.344251
40	2.345087
51	2.738930
99	2.849246
137	2.850217
145	2.850227

Örnek 3.3 Gezgin Satıcı Problemi (Carr 2014)

Bu belki de en meşhur, pratik ve geniş çapta incelenen kombinatoryal optimizasyon problemidir (kombinatoryal derken bunun permütasyonları - farklı sıralarda bir dizi öğeyi düzenlemeyi içerdiği kastedilir). Robotik, devre kartı delme, kaynak, imalat, nakliye ve diğer birçok alanda uygulamaları vardır. Örneğin, bir devre kartında on binlerce delik olabilir ve bir matkabın bazı maliyet fonksiyonlarını (örneğin zaman veya enerji) en aza indirirken bu deliklerin her birini ziyaret edecek şekilde programlanması gerekir.

Bu problemde, mümkün olan en kısa yoldan n şehri ziyaret etmesi gereken gezgin bir satıcı olduğu varsayılmaktadır. Her şehri tam olarak bir kez ziyaret etmekte, sonra

başladığı şehre geri dönmektedir. Bu nedenle, bir çözüm adayı, tüm şehirlerin ziyaret sırasına göre bir listesi olacaktır. Şehirler şehir 1, şehir 2, şehir 3, ... , şehir n olarak adlandırılmıştır ve şehir 1 başlangıç noktasıdır. Bu nedenle, genetik algoritma için kromozomlar, 1'den n'ye kadar tam sayıların farklı permütasyonları olacaktır.

Gezgin satıcı probleminin birçok çeşidi vardır, ancak amaçlar için aşağıdaki varsayımlar yapılmaktadır. Tüm $i \in [1,n]$ ve $j \in [1,n]$ için c_i şehrinden c_j şehrine olan d mesafesinin $d(c_i,c_j) = d(c_j,c_i)$ olduğu varsayılmaktadır. Bazı gerçek dünyadaki durumlarda bunun nasıl doğru olmayacağı görülebilir: bir yöndeki otoyol, ters yönde giden otoyoldan biraz daha uzun olabilir veya yokuş yukarı gitmek yokuş aşağı gitmekten daha pahalı olabilir (seyahat maliyetinin mesafeye dahil edilmesi yaygındır). Bu durumlara "asimetrik" seyyar satıcı problemleri adı verilir. Satıcı başladığı şehirde bitirdiğinden, bu "kapalı" bir seyyar satıcı problemidir. "Açık" varyasyonda, satıcı tüm şehirleri tam olarak bir kez ziyaret eder, böylece yolculuğu başladığı yerde bitirmez.

Permütasyon Problemleri için Çaprazlama ve Mutasyonu Değiştirme

Aday çözümlerin kromozomlara nasıl kodlandığına bağlı olarak, ikili genetik algoritma için tanımlanan şekliyle çaprazlama ve mutasyon operatörleri çalışmayacaktır çünkü her şehrin bir kez ve yalnızca bir kez temsil edilmesi gerekir.

Bunun neden böyle olduğunu anlamak için, 5 uzunluğundaki iki ebeveyn kromozomu ve normal çaprazlamadan sonraki yavrularını göz önüne alalım:

Ebeveynler	Normal Çaprazlama	Yavru(hatalı)
35124	35|124	35532
14532	14|532	14124

Bu yavrular, 1'den 5'e kadar olan tam sayıların geçerli permütasyonları değildir çünkü bazı rakamları eksiktir ve diğerlerini tekrar eder. Bu engeli aşmak için, bu genetik algoritmada "döngü" çaprazlama adı verilen bir teknik kullanılacaktır. Goldberg (1989) ve Haupt'un (2004) çalışmalarında tartışılan permütasyonlara uyum sağlamak için çaprazlamayı değiştirmenin birkaç diğer yolu verilmiştir. Döngü çaprazlama, Çizelge 3.5'te 6 uzunluğundaki kromozomlarla gösterilmektedir.

Çizelge 3.5 Döngü çaprazlama örneği

Ebeveynler	Yavru (adım 1)	Yavru (adım 2)	Yavru (adım 3)	Yavru (adım 4)
415|3|26	4152|2|6	4|1|5216	|4|45216	**34**5**2**16
346|2|15	3463|1|5	3|4|6325	|3|16325	**41**63**2**5

Önce kromozomun uzunluğunda rastgele bir yer seçilir. İki ebeveyn kromozom, yavruları oluşturmak için bu konumda (Çizelge 3.5'te çubuklarla işaretlenmiş) tamsayılar yer değiştirir. Değiştirilen tamsayılar aynı değere sahip olmadıkça, her yavru, eş bir tam sayıya sahiptir. Daha sonra, ilk yavrudaki yinelenen tamsayıyı, ikinci yavruda aynı konumda olanla değiştirilir. Bu, şimdi farklı bir tamsayının bir kopyası olduğu anlamına gelir, bu nedenle süreç, ilk yavruda (veya ikinci yavru) kopya kalmayana kadar tekrar eder. Şimdi her yavru, tam olarak her tam sayıdan birine sahiptir, dolayısıyla her ikisi de geçerli permütasyonlardır.

Değiştirilmiş mutasyon operatörü, yeni nesilden bir kromozomda rastgele iki tam sayı seçer ve bunları değiştirir. Bu operatör hala tipik olarak düşük bir olasılıkla gerçekleşir.

Genetik Algoritma Aşaması

Bu bölümde, gezgin satıcı problemini çözmek için genetik algoritma oluşturulacaktır. Bunu yapmak için, permütasyon genetik algoritması ve bir gezgin satıcı probleminin uygunluk fonksiyonu için Haupt'un kodu uyarlanıp ve geliştirilmiştir. Bu algoritma için MATLAB kodu EK 1'de verilmektedir.

20 şehir olduğunu varsayalım. Bunları temsil etmek için, xy-düzleminde (0,0) ve (1,1) arasındaki 1x1 karede rastgele 20 nokta yerleştirilir. Daha önce, kromozomlar 1'den 20'ye kadar olan tam sayıların permütasyonları olarak tanımlanmıştı.

N şehrin verilen kromozomda $c_1 \to c_2 \to ... \to c_n$ sırasına göre listelendiğini varsayalım. Gezgin satıcıların kat ettiği toplam mesafe en aza indirmeye çalışıldığı için, uygunluk fonksiyonu toplam mesafe D:

$$D = \sum_{k=1}^{n} d(c_k, c_{k+1})$$

olacaktır. Burada şehir n+1=şehir 1 (başlangıç şehri) dir. Eğer (x_i, y_i)'nin şehir c_i'nin koordinatlarını gösterdiği varsayılırsa, bu durumda uygunluk fonksiyonu

$$D = \sum_{k=1}^{n} \sqrt{(x_{k+1} - x_k)^2 + (y_{k+1} - y_k)^2}$$

olur.

Daha sonra, popülasyon büyüklüğü 20 olarak tanımlanır. Bu algoritmayla, sonraki nesilleri inşa etmek için önceki örneklerden farklı bir yaklaşım kullanılır. Tüm popülasyonu yeni yavrularla değiştirmek yerine, mevcut nüfusun uygun yarısı elde tutulur ve yeni neslin diğer yarısı seçim ve çaprazlama yoluyla üretilir. Seçim operatörü de bu durumda yalnızca tutulan popülasyon oranından seçim yapacaktır (Bkz. Şekil 3.2). Bu algoritmada, mutasyon operatörü, 0.05 olasılıkla yeni neslin her bir üyesine etki etmektedir.

```
for ii =2: keep
odds =[ odds ii* ones (1, ii )];
end
Nodds = length ( odds );
```

Şekil 3.2 En iyi 10 kromozomun uygunluk sıralamasına göre seçim operatörü için olasılıkların belirlenmesi

Mevcut nesil kromozomlar en yüksek uygunluktan en düşük uygunluğa (1'den 20'ye kadar) sıralanır ve yalnızca en uygun yarısı (10 kromozom) tutulur. Şekil 3.2'deki odds değişkeni, 1'den 10'a kadar tamsayıları içeren bir vektördür; 1 tamsayısının on örneği, 2 tamsayısının dokuz örneği, 10 tamsayısının bir örneğine kadar sıralanır. Daha sonra bir kromozom, vektör olasılıklarından rastgele bir tamsayı seçilerek ebeveyn olarak seçilir. Bu nedenle, seçilen en uygun kromozomun olasılığının 10/55 = 2/11 olduğu ve en az uygun kromozomun seçilme olasılığının 1/55 olduğu görülebilir.

Son olarak, yineleme sayısı 10.000 olarak belirlenmiştir. Şekil 3.3-3.6, birkaç çalışmadan dördünü göstermektedir. Optimal çözümün ne olduğu önceden bilinmediği için, en iyi sonuç tatmin edici sayıda tekrarla görünene kadar tüm algoritmayı birçok kez çalıştırmak önemlidir. Algoritma 4.1211 sonucunu görecek şekilde yeterince çalıştırılmamış olsaydı, Şekil 3.4 ve Şekil 3.6'dan, 4.1816'nın mümkün olan en kısa mesafe olduğu kanaatine varılabilirdi. Şekil 3.3'de 4,2547 mesafe ile en iyi çözüm:

[13; 15; 4; 18; 11; 17; 10; 14; 1; 3; 19; 12; 5; 20; 8; 2; 9; 7; 16; 6],

Şekil 3.4'de 4,1816 mesafe ile en iyi çözüm:

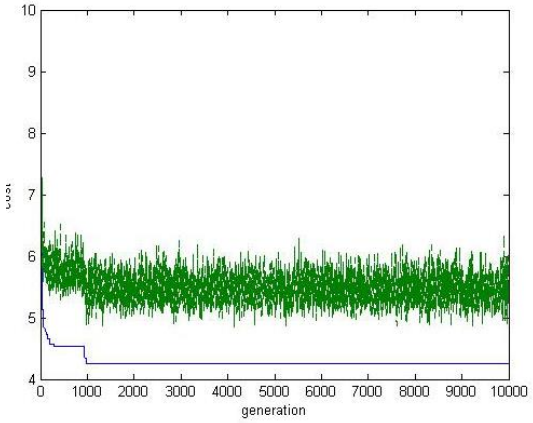
[4; 18; 11; 17; 10; 14; 1; 3; 19; 2; 9; 16; 7; 8; 12; 5; 20; 6; 13; 15],

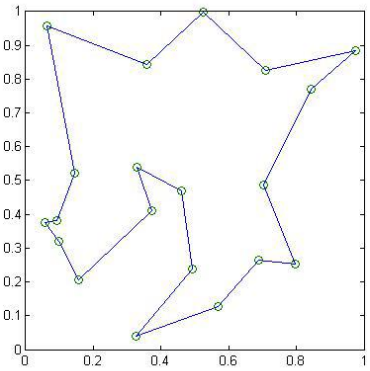
Şekil 3.5'de 4.1211 mesafe ile en iyi çözüm:

[7; 9; 8; 2; 19; 3; 1; 14; 10; 12; 20; 5; 17; 11; 18; 4; 15; 13; 6; 16] ve son olarak

Şekil 3.6'da 4.1816 mesafe ile en iyi çözüm:

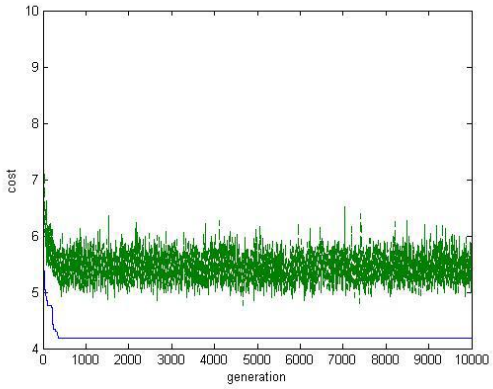
[10; 14; 1; 3; 19; 2; 9; 16; 7; 8; 12; 5; 20; 6; 13; 15; 4; 18; 11; 17] olarak elde edilmiştir

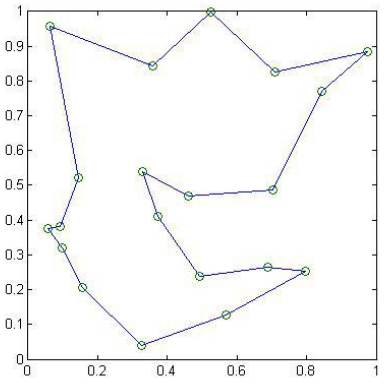




(a) (b)

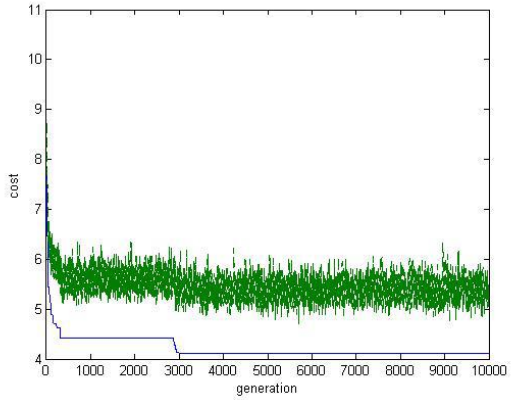
Şekil 3.3 (a) Mavi eğri en kısa mesafe, yeşil eğri ortalama mesafedir. (b) Rotanın grafiği.
Kapalı bir döngüde 20 şehre seyahat mesafesini en aza indiren bir genetik algoritmanın ilk çalışması

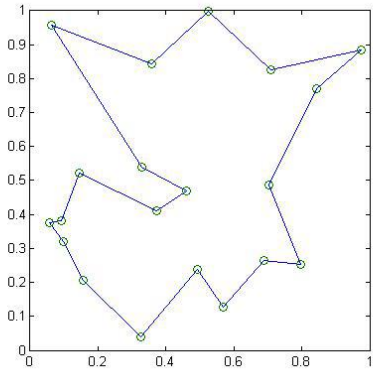




(a) (b)

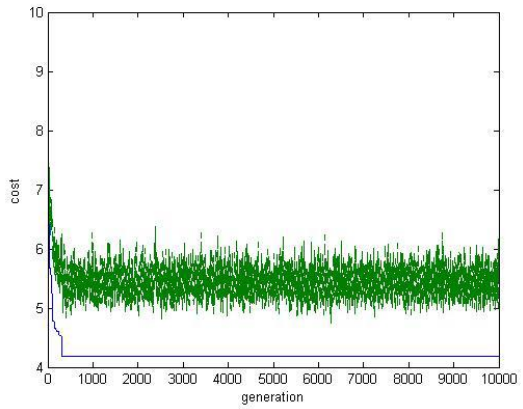
Şekil 3.4 Mavi eğri en kısa mesafe, yeşil eğri ortalama mesafe, (b) Rotanın grafiği. Kapalı bir döngüde 20 şehre seyahat mesafesini en aza indiren bir genetik algoritmanın ikinci çalışması

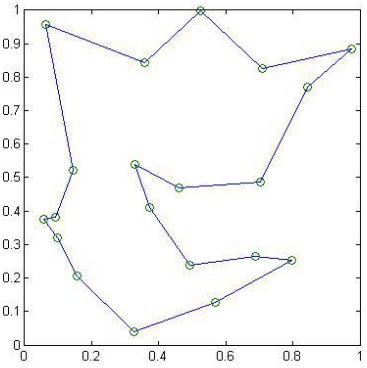




(a) (b)

Şekil 3.5 (a) Mavi eğri en kısa mesafe, yeşil eğri ortalama mesafe (b) Rotanın grafiği. Kapalı bir döngüde 20 şehre seyahat mesafesini en aza indiren bir genetik algoritmanın üçüncü çalışması





(a) (b)

Şekil 3.6 Mavi eğri en kısa mesafe, yeşil eğri ortalama mesafe, (b) Rotanın grafiği Kapalı bir döngüde 20 şehre seyahat mesafesini en aza indiren bir genetik algoritmanın dördüncü çalışması.

4.1211'den daha kısa bir mesafenin olmadığı birkaç çalıştırmada doğrulanmıştır, bu nedenle en uygun çözüm şu şekildedir:

[7; 9; 8; 2; 19; 3; 1; 14; 10; 12; 20; 5; 17; 11; 18; 4; 15; 13; 6; 16].

Bunu onaylayan çalışmalardan birinde, kromozom farklı bir permütasyon olsa bile, bu çalıştırmanın bulduğu optimum çözümün Şekil 3.5'deki çalıştırmanın çözümüne eşdeğer bir çözüm olduğuna dikkat edilmiştir ([20; 12; 10; 14; 1; 3; 19; 2; 8; 9; 7; 16; 6; 13; 15; 4; 18; 11; 17; 5]). Bu kromozom farklı bir şehirde başlamıştır, ancak bu,

satıcının yolu kapalı bir döngü olduğu için mesafeyi etkilememektedir. Bu kromozom ayrıca şehirleri 3. çalıştırma sırasının tersi sırayla listeler, ancak şu hatırlanmalıdır ki c_i, c_j şehirleri için

$$d\left(c_i, c_j\right) = d\left(c_j, c_i\right)$$

dir, yani bu da mesafeyi etkilemez.

Örnek 3.4 Sırt Çantası Problemi (Steeb 2001)

Burada, kriptografide kullanılan, hammadde toplamanın en iyi yolunu ve diğer birçok uygulamayı belirleyen bir başka klasik optimizasyon problemi sunulmaktadır. Bu sefer problemi uygunluk fonksiyonuna ve kromozomlara çevirmek için sadece olası bir yöntem ve seçim operatörü için olasılıkları dağıtmanın yeni bir yolu sunulacaktır. Bu örnek Steeb'in (2001) çalışma kitabından uyarlanmıştır.

20 kg'dan fazlasını taşıyamayacak sırt çantalı bir gezi planlayan bir yürüyüşçü var olduğunu kabul edelim. Yürüyüşçü, yanında götürmek istediği tüm malzemeleri bir kez seçtikten sonra, bunların hepsinin 20 kg'ı aştığını görecektir. Ardından, hangisini alacağını seçmesine yardımcı olması için Çizelge 3.6'da gösterilen 12 öğenin her birine bir değer atar. 20 kg'ı geçmeden taşıdıklarının değerini en üst düzeye çıkaracak şekilde, yürüyüşçü yanına hangi eşyaları almalıdır?

Öğeleri kromozomlara çevirmenin bir yolu, her bit konumunun belirli bir öğeyi temsil ettiği 12 uzunluğunda bit dizileri oluşturmaktır. 1 belirli bir öğenin dahil edildiğini, 0 ise onun hariç tutulduğunu göstersin.

Çizelge 3.6 Kamp malzemelerinin değeri ve ağırlığı

Ürün	Değer	Ağırlık
Böcek kovucu	12	2
Kamp ocağı	5	4
Kantin (dolu)	10	7
Giysi	11	5
Kurutulmuş yemek	50	3
İlk yardım çantası	15	3
El feneri	6	2
Roman	4	2
Yağmur dişli	5	2
Uyku tulumu	25	3
Çadır	20	11
Su filtresi	30	1

Buna dayanarak, kromozomlar bu bilgiyi taşımadığından, uygunluk fonksiyonu her bir bit pozisyonuyla ilişkili sabit ağırlıkları ve değerleri bilmelidir. Uygunluk fonksiyonu aynı anda, uygunluğu belirlemek için öğelerin değerlerinin toplamı ve ağırlık sınırını karşılamak için öğelerin ağırlıklarının toplamı olmak üzere, iki toplamı toplayabilmelidir. Uygunluk fonksiyonu değerleri ve ağırlıkları toplarken, kromozomların ağırlık sınırı içinde kalmasını sağlamanın bir yolu, toplamayı hemen durdurmak ve eğer bir kromozomun ağırlığı 20 kg'ı aşarsa 0 veya -1'lik bir uygunluk atamaktır. Bu problemin R kodu EK 2'de verilmiştir.

4. SONUÇ

Bu tezde optimizasyonda evrimsel hesaplamaların önemli bir alt sınıfı olan Genetik Algoritmalar kavramsal olarak incelenmiş ve bu yöntemin çeşitli problemlere olan uygulamalarına değinilmiştir. Genetik Algoritmaların önemi bilgisayar teknolojisindeki ilerlemeler sayesinde gün geçtikçe hız kazanmış ve Genetik Algoritmalar günümüzde klasik optimizasyon yöntemleriyle çözülemeyen birçok karmaşık problemin çözümünde hızlı ve güvenilir bir yöntem olarak yerini almıştır. Evrim teorisinden ve genetikten ilham alan güçlü bir optimizasyon tekniği olan Genetik Algoritmalar üzerindeki çalışmalar gün geçtikçe artmaktadır. Özellikle son zamanlarda diferansiyel denklemler üzerinde yapılan Genetik Algoritma çalışmalarında artış görünmektedir ve bu sayede birçok gerçek hayat modeli yeni bir bakış açısıyla yeniden yorumlanabilecektir.

KAYNAKLAR

Carr, J. 2014. An Introduction to Genetic Algorithms. Web sitesi: https://www.whitman.edu/Documents/Academics/Mathematics/2014/carrjk.pdf. Erişim tarihi: 20.06.2021.

Emel, G.G. ve Taşkın, Ç. 2002. Genetik Algoritmalar ve Uygulama Alanları, Uludağ Üniversitesi İktisadi ve İdari Bilimler Fakültesi Dergisi, Cilt XXI, Sayı 1, s.129-152.

Gen, M. and Cheng, R. 1999. Genetic Algortihms and Engineering Optimization, Wiley-Interscience, New York.

Goldberg, D.E. 1989. Genetic Algorithms in Search Optimization and Machine Learning, Addison –Wesley Longman.

Haupt, R.L. and Haupt, S.E. 2004. Practical Genetic Algorithms, John Wiley and Sons, New York.

Holland, J. 1975. Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, MIT Press, Cambridge, MA, 1992.

Koza, J.R. 1992. Genetic Programming, MIT Press, Cambridge, MA.

Mateescu, G.D. 2006. On The Application of Genetic Algorithms to Differential Equations. Romanian Journal of Economic Forecasting, 7(2); 5-9.

McCall J. 2005. Genetic algorithms for modelling and optimisation. Journal of Computational and Applied Mathematics. 184, 205–222.

Michalewicz, Z. 1992. Genetic Algorithms + Data Structure = Evolution Programs, Springer & Verlag, Berlin.

Mitchell, M. 1999.An Introduction to Genetic Algorithms, MIT Press, Fifth Printing.

Özturan, A.T. 2019. Optimizasyon ve Matlab Uygulamaları, Nobel Akademik Yayıncılık, Ankara.

Pedregal P. 2004. Introduction to Optimization, Springer-Verlag New York, Inc.

Satman, M.H. 2016. Genetik Algoritmalar, Türkmen Kitabevi, İstanbul.

Simon, D. 2013. Evolutionary Optimization Algorithms, John Wiley & Sons.

Sivanandam, S. N. and Deepa, S.N. 2008. Introduction to Genetic Algorithms, New York.

Steeb, W. 2001. The Nonlinear Workbook : Chaos, Fractals, Cellular Automata, Neural Networks, Genetic Algorithms, Fuzzy Logic: with C++, Java, SymbolicC++ and Reduce Programs. Singapore: World Scientific.

EKLER

EK 1 Gezgin Satıcı Problemi MATLAB Kodu

Definition of Fitness Function:

```
function dist=tspfun(pop)
% Cost function for the traveling salesman problem.
% Uses global variables "x" and "y"
%
% Haupt and Haupt, 2003
% Edited and modified by Hundley and Carr, 2014
global x y
[Npop,Ncity]=size(pop);
tour=[pop pop(:,1)];
%distance between cities
for ic=1:Ncity
for id=1:Ncity
dcity(ic,id)=sqrt((x(ic)-x(id))^2+(y(ic)-y(id))^2);
end % id
end % ic
% cost of each chromosome
for ic=1:Npop
dist(ic,1)=0;
for id=1:Ncity
dist(ic,1)=dist(ic)+dcity(tour(ic,id),tour(ic,id+1));
end % id
end % ic
```

Main Algorithm:

```
%% Genetic Algorithm for permutation problems
% minimizes the objective function designated in ff
%
% Haupt and Haupt, 2003
% Edited and modified by Hundley and Carr, 2014
clear
global iga x y
%% Setup the GA
ff='tspfun'; % filename objective function
npar=20; % # of optimization variables
Nt=npar; % # of columns in population matrix
x=rand(1,npar);
y=rand(1,npar); % cities are at (xcity,ycity)
% Uncomment next line to use the same set of cities in multiple runs
% load cities0
% Stopping criteria
maxit=10000; % max number of iterations
% GA parameters
popsize=20; % set population size
mutrate=.05; % set mutation rate
selection=0.5; % fraction of population kept
keep=floor(selection*popsize); % #population members that survive
M=ceil((popsize-keep)/2); % number of matings
odds=1;
for ii=2:keep
odds=[odds ii*ones(1,ii)];
end
Nodds=length(odds);
odds=keep-odds+1; % odds determines probabilities for being parents
% Create the initial population
```

```
iga=0; % generation counter initialized
for iz=1:popsize
pop(iz,:)=randperm(npar); % random population
end
cost=feval(ff,pop); % calculates population cost using ff
[cost,ind]=sort(cost); % min cost in element 1
pop=pop(ind,:); % sort population with lowest cost first
minc(1)=min(cost); % minc contains min of population
meanc(1)=mean(cost); % meanc contains mean of population
%% Iterate through generations (MAIN LOOP)
while iga<maxit
iga=iga+1; % increments generation counter
% Pair and mate
pick1=ceil(Nodds*rand(1,M)); % mate #1
pick2=ceil(Nodds*rand(1,M)); % mate #2
% ma and pa contain the indices of the parents
ma=odds(pick1);
pa=odds(pick2);
% Performs mating
for ic=1:M
mate1=pop(ma(ic),:);
mate2=pop(pa(ic),:);
indx=2*(ic-1)+1; % odd numbers starting at 1
xp=ceil(rand*npar); % random value between 1 and N
temp=mate1;
x0=xp;
while mate1(xp)~=temp(x0)
mate1(xp)=mate2(xp);
mate2(xp)=temp(xp);
xs=find(temp==mate1(xp));
xp=xs;
```

```
end
pop(keep+indx,:)=mate1;
pop(keep+indx+1,:)=mate2;
end
% Mutate the population
nmut=ceil(popsize*npar*mutrate);
for ic = 1:nmut
row1=ceil(rand*(popsize-1))+1;
col1=ceil(rand*npar);
col2=ceil(rand*npar);
temp=pop(row1,col1);
pop(row1,col1)=pop(row1,col2);
pop(row1,col2)=temp;
im(ic)=row1;
end
cost=feval(ff,pop);
%_____________________________________________________________
% Sort the costs and associated parameters
part=pop;
costt=cost;
[cost,ind]=sort(cost);
pop=pop(ind,:);
%_____________________________________________________________
% Do statistics
minc(iga)=min(cost);
meanc(iga)=mean(cost);
end %iga
%_____________________________________________________________
% Displays the output
day=clock;
disp(datestr(datenum(day(1),day(2),day(3),day(4),day(5),day(6)),0))
```

```
disp(['optimized function is ' ff])
format short g
disp(['popsize=' num2str(popsize) 'mutrate=' num2str(mutrate) '# par=' num2str(npar)])
disp([' best cost=' num2str(cost(1))])
disp(['best solution']); disp([num2str(pop(1,:))])
figure(2)
iters=1:maxit;
plot(iters,minc,iters,meanc,'--');
xlabel('generation');
ylabel('cost');
figure(1);
plot([x(pop(1,:)) x(pop(1,1))],[y(pop(1,:)) y(pop(1,1))],x,y,'o');
axis square
```

EK 2 Sırt Çantası Problemi R Kodu

```
require("GA")
c.params<-c(12,5,10,11,50,15,6,4,5,25,20,30)
a.params<-c(2,4,7,5,3,3,2,2,2,3,11,1)
bound<-20

f<-function(bits){
  goal<-sum(c.params*bits)
  constraint<-sum(a.params*bits)
  cost<-goal
  if(constraint > bound){
    cost<-cost-12*goal
  }
  return(cost)
}

myga<-ga(type="binary",
         nBits=length(c.params),
         popSize = 100,
         maxiter = 200,
         fitness = f
)
print("Cozum:")
print(myga@solution)
print("Toplam Fayda:")
print(sum(c.params*myga@solution))
print("Toplam Agirlik:")
print(sum(a.params*myga@solution))
```