KÜTAHYA SAĞLIK BİLİMLERİ ÜNİVERSİTESİ
MÜHENDİSLİK VE DOĞA BÖLÜMLERİ FAKÜLTESİ

YAPAY ZEKA DERSİ

NÖBET ÇİZELGELEME PROBLEMLERİNİN GENETİK ALGORİTMALARLA OPTİMİZASYONU

Final Rapor

Mustafa AKER

2118121039

14.06.2024

Özet-Abstract

Bu çalışmada hemşire çizelgeleme problemi ele alınmıştır. Çizelgeleme çalışmaları bir çok alanda yaygın olarak kullanılır. İşçilerin ve makinaların çalışma aralıkalrı ,ders programlarının oluşturulması veya uçakların kalkış iniş saatlerinin belirlenmesinde gibi bir çok alanda cizelgeleme yapılmaktadır. Ancak sağlık sektöründeki önemi de oldukça büyüktür.. Çizelgelerin elle oluşturulması sorumlu olan kişilerin 6-8 saat kadar zamanını aldığı aşıkardır . Bu koşullar altında bu süreyi minimuma indirecek bir bilgisayar uygulaması şarttır. Çizelgeleme problemleri, çözümü zor olan problemler grubuna girmektedir. Bu nedenle yaklaşık en iyi çözüm değerlerine ulaşabilmek için genetik algoritmalar kullanılmıştır. Çalışma, Buca Seyfi Demirsoy Devlet Hastanesi'nden alınan gerçek verilerle

yapılmıştır. Hemşirelerin en uygun çalışma saatlerini bulmak için oluşturulan modelin çözümünde Python programlama dilinde genetik algoritmalar aracından yararlanılmıştır. Son olarak elde edilen sonuçlar test edilmiş ve tablo şeklinde sunularak çalışma tamamlanmıştır.

Anahtar Kelimeler: Optimizasyon, Genetik Algoritma, Hemşire Çizelgeleme.

1 Giriş

Sağlık hizmetlerinin temel amacı; kişi, aile ve toplumların sağlıklarının korunması, geliştirilmesi, hasta olanların tedavi edilmesi ve tedavi edilenlerin geri kalan yaşamlarını sağlıklı olarak sürdürülebilmelerini sağlamaktır. İnsanların sağlık hizmetlerinden yeterince, yerinde, zamanında ve gereksiz masraflardan kaçınarak yararlanmaları önemlidir. Ancak bir kurumun 7 gün 24 saat açık olarak hizmet vermesi gerçekten taktiredilmesi gereken bir çalışmadır. Personelin doğru atanabilmesi büyük bir planlama gerektirir. İşte tüm bu ihtiyaçlar bu çalışmada çizelgelemenin tercih edilmesine sebep olmuştur. .

Çizelge (schedule); çizgilerle bölümlere ayrılmış kâğıt anlamına gelir. Çizelge yardımıyla; kadro, kademe, basamak ve derecelerin yer aldığı bir liste elde edilebilir. Çizelgeleme de eldeki işlerin bir grup kaynağa atanmasıdır.[1]

Çizelgeleme problemleri; en düşük işgücü maliyetleri ile çalışma vardiyalarına atanmaların belirlenmesini, personel hizmet kalitesi ve çalışma yasalarıyla ilgili kısıtlamaların karşılanmasını konu alan problemlerdir.[2]

Çizelgelemedeki asıl amaç, daha az kaynakla ve daha az sürede, istenilen kriterlere uyacak biçimde problemin çözümüne ulaşmaktır. Aslında günlük hayatta bir şekilde kişisel not defteri kullanarak yaptığımız planlar da bir çizelge oluşturmaktır. Ancak elbette bir okul, hastane ya da fabrika söz konusu olduğunda değerler büyüyecek ve değişken sayısı artacaktır.Kurumların her türlü unsuru göz önünde bulundurması ve uygun bir çizelge oluşturabilmesi gerekir. Bu sebeple daha kısa sürede bu problemi çözmemizi sağlayacak yollar aranacaktır.

Sezgisel algoritmalar genellikle en iyiye yakın olan çözüm yoluna hızlı ve kolay bir şekilde ulaştıklarından bu çalışmada sezgisel algoritmalardan olan Genetik Algoritma kullanılmıştır.

2 Literatür Araştırması

[1] İnanç ve Şenaras (2020), evde bakımda hemşire çizelgeleme problemi için Genetik Algoritma kullanmışlardır.

[2] Küçük ve Deveci Kocakoç (2020) hemşirelerin en uygun çalışma saatlerini bulmak icin MATLAB programının GA aracından yararlanılmıştır

[3] 90'ların sonlarında sezgisel arama yöntemleri hemşire çizelgeleme için de kullanılmaya başlanmıştır. Yamamura ve diğerlerinin (1993) çalışmaları, GA ile yapılan ilk hemşire çizelgeleme uygulamasıdır.

[4] Bailey ve diğerleri (1997), farklı beceri seviyelerine sahip personelin planlanması sorununa Benzetilmiş Tavlama ve GA uygulamıştır. Sonuçlar, optimal veya optimale yakın çözümler üretilebileceğini göstermektedir.

[5] Aikelin ve diğerlerinin (2004), ele aldıkları yaklaşım, hemşirelerin permütasyonlarına dayanan dolaylı bir kodlama ile programlar oluşturan sezgisel bir kod çözücü kullanmaktır. Sonuçlar, önerilen algoritmanın yüksek kaliteli çözümler sunabildiğini ve yakın zamanda yayınlanan bir Tabu Arama yaklaşımından daha hızlı ve daha esnek olduğunu ortaya koymaktadır.

[6] Duenas ve diğerleri (2009), hemşirelerin tercihlerini içeren çok amaçlı bir çizelgeyi ele almaktadır. Bu tercihler bulanık kümelerle modellenmiştir ve GA ile birleştirilmiş melez bir çözüm yöntemi kullanılmıştır. Sonuçlar, önerilen yaklaşımın iyi kalitede çözümler ürettiğini ve gerçek hayattaki problemlere uygulanabilir olduğunu göstermektedir.

[7] Balekar ve Mhetre (2013), hemşire çizelgeleme için GA yaklaşımının kullanıldığı çalışmaları toparlayan bir kaynakça sunmuştur.

[8] Leksakul ve Phetsawat çalışmasında elde edilen sonuçlara göre (2014) GA tarafından bulunan hemşire programı,mevcut programla karşılaştırıldığında aylık personel giderlerinde % 12 ve hemşire sayısında % 13 tasarruf görülmüştür.

[9] Kim ve diğerleri (2018), popülasyon büyüklüğü ve mutasyon oranı parametrelerini dikkate alarak Memetik Algıritma ve GA kıyaslamışlardır.

[10] Alfadilla ve diğerleri (2019), acil serviste hemşire çizelgelemesi ile ilgilenmişlerdir. GA hemşirelerin yerine getirilmemiş tercihlerini en aza indirmek için kullanılmaktadır.

3 Metodoloji

Sezgisel algoritmalar genellikle en iyiye yakın olan çözüm yoluna hızlı ve kolay şekilde ulaştıklarından, çalışmada sezgisel algoritmalardan biri olan Genetik Algoritma (GA) kullanılacaktır. GA, tek çözüm değil birden fazla optimal çözüm elde etmesi, çok sayıda parametre ile çalışma imkanı olması, amaç fonksiyonunu geniş bir açıda araştırması nedeniyle yararlı bir yöntemdir.

3.1 Genetik Algoritma

Evrimsel hesaplamanın bir alt dalı olan genetik algoritmalar, yapay zekânın hızla gelişen alanlarından biridir. Bu metasezigel yaklaşımların esin kaynağı Darwin'in evrim teorisidir. Evrimsel hesaplama kavramı, 1960'lı yıllarda I.Rechenberg'in "Evrim Stratejileri" adlı çalışmasında ortaya atılmıştır. Takip eden 1970'li yıllarda da Michigan Üniversitesi'nden John Holand tarafından genetik algoritmalar bulunmuş ve öğrencileri ile meslektaşları yardımıyla geliştirilmiştir. Bunu 1975 yılında Holland'ın "Doğal ve Yapay Sistemlerde Uyum" adlı kitabını yayınlaması izlemiştir.

Genetik algoritmalar konusundaki esas gelişim ise, John Holland'ın doktora öğrencisi David E. Goldberg tarafından 1985 yılında hazırlanan "Gaz ve Boru Hatlarının Genetik Algoritma Kullanılarak Denetlenmesi" konulu doktora tezi ile sağlanmıştır. Ulusal Bilim Fonu tarafından verilen Genç Araştırmacı Ödülü'nü kazanan Goldberg, dört yıl sonra 1989 yılında yayımladığı "Makine Öğrenmesi, Arama ve Optimizasyon İçin Genetik Algoritma" adlı kitabı ile genetik algoritmaya yeni bir boyut kazandırmıştır. Bu çalışma günümüzde dahi genetik algoritma konusunda en kapsamlı referans olma özelliğindedir.

Genetik algoritma yöntemi, evrim teorisi esaslarına göre çalışarak, verilen bir sorun için en iyi çözüm veya çözümleri bulmaya yarar. Bu arayışı, karar değişkeni uzayındaki birçok başlangıç noktasından başlayarak, paralel işlemler dizisi ile en iyi yöne doğru gelişerek yapar. Karar uzayındaki bu noktalarda uygunluk derecelerinden başka bilgilere gerek yoktur. Toplumdaki noktaların paralel çalışarak en iyiye doğru gelişmesi rastgelelik ilkeleri ile sağlanmaktadır. Genetik algoritmanın esası doğal seçme ve genetik kurallarına dayanmaktadır. Bu kurallar ortama en fazla uyum sağlayan canlıların hayata devam etmesi ve uyum sağlayamayanların da elenmesi olarak algılanmalıdır.

Genetik algoritmalar doğadaki en iyinin yaşamasını gerektirmekte ve bunu belirleyen uygunluk işlevi, yeni çözümler üretmek için çaprazlama,

kopyalama ve değiştirme gibi operatörleri kullanmaktadır. Genetik algoritmanın önemli özelliklerinden birisi de bir grup üzerinde çözümü araması ve bu sayede çok sayıda çözümün içinden en iyiyi seçmesidir.

Probleme ait en iyi çözümün bulunabilmesi için;

1. Bireylerin gösterimi doğru bir şekilde yapılmalı,
2. Uygunluk fonksiyonu etkin bir şekilde oluşturulmalı,
3. Doğru genetik işlemciler seçilmelidir,

3.2 Genetik Algoritmaların Diğer Yöntemlerden Farkı

Goldberg'e göre genetik algoritmayı diğer arama yöntemlerinden ayıran en belirgin özellikleri çözüm arama şeklinin farklı oluşudur.

1. Genetik algoritma, parametre setlerinin kodları ile ilgilenir, parametrelerin kendileri ile doğrudan ilgilenmez,
2. Genetik algoritmanın arama alanı, yığının veya populasyonun tamamıdır; tek nokta veya noktalarda arama yapmaz,
3. Genetik algoritmalarda, amaç fonksiyonu kullanılır, sapma değerleri veya diğer hata faktörleri kullanılmaz,
4. Genetik algoritmaların uygulanmasında kullanılan operatörler, stokastik yöntemlere dayanır, deterministik yöntemler kullanılmaz

Başka bir ifadeyle genetik algoritmalar doğal olayların gelişmesindeki genetik mekanik ilkelere göre çalışırlar.

- Genetik algoritmalar, çözümlemesinde karar değişkenlerini genetik sayı sistemine göre kodlayarak kullanır. Sayı sisteminde karar değişkenlerinin genleri topluca karar uzayında bir noktayı temsil eder.
- Genetik algoritmalarda bir nokta yerine aynı anda noktalar topluluğundan hareket edilir. Topluluğun genetik algoritmalar evrimi ile gelişmesi sonucunda en iyi çözüme ulaşılır. Evrim sırasında sistem yerel en iyiye takılmaz
- Genetik algoritmalar evrimi sırasında, karar değişkenlerinin belirttiği noktalardaki hedef fonksiyonu değerleri kullanılır. Türev ve integral işlemlerine gerek olmadığından başlangıç ve sınır şartları ile bazı klasik kabullerin yapılmasına gerek yoktur

- Genetik algoritmalar evrim işlemleri belirlilik değil kurallarına dayanır. Seçim işlemleri ihtimal ilkeleri ışığı altında yapılır. En iyi için sayılan klasik yöntemlerin dışında daha basit, fazla matematik gerektirmeyen ve objektif olan GA (genetik algoritma) yaklaşımları kullanılabilir. Genetik algoritmalardaki rastgele yaklaşımlar sonuç çözümün yerel en iyi çözümlere takılıp kalmamasını sağlar

3.3 Temel Kavramlar

[2] Genetik algoritmanın çalışmasında ve başarılı çözüm değerlerine ulaşılmasında temel kavramların iyi anlaşılmasının ve belirlenmisinin önemi büyüktür.

3.3.1 Gen

Kalıtsal molekülde bulunan ve organizmanın karakterlerinin tayininde rol oynayan kalıtsal birimlere denir. Yapay sistemlerde gen, kendi başına anlamlı birer genetik bilgi taşıyan en küçük yapı birimi olarak alınır.

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

3.3.2 Kromozom

Bir ya da birden fazla gen yapısının bir araya gelerek oluşturduğu problemin çözümüne ait tüm bilgiyi içeren diziye kromozom denir. Kromozomlar, alternatif aday çözümleri gösterirler

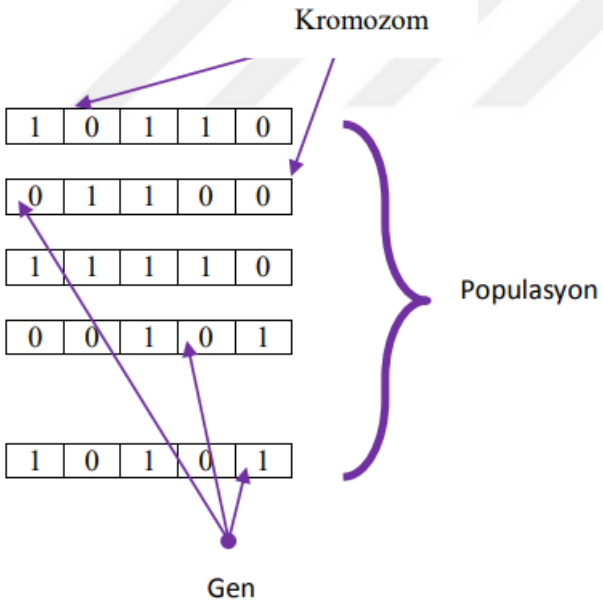
Kromozornlann bir araya gelmesiyle populasyon (yığın) oluşturulur. Yığındaki her bir bireye kromozom, kromozomdaki her bir bilgiye gen denir. Kromozomlar, üzerinde durulan problemin olası çözüm bilgilerini içermekledir Kromozom, GA yaklaşımında üzerinde durulan en önemli birim olduğu için bilgisayar ortamında iyi ifade edilmesi gerekir.

3.3.3 Populasyon

Populasyon, çözüm bilgilerini içeren kromozomların bir araya gelmesiyle oluşan olası çözüm yığınına denir. Yığındaki kromozoın sayısı sabit olup problemin özelliğine göre programlayıcı tarafından belirlenir .

Çözüm uzayı açısından populasyon büyüklüğünün küçük değerde olması çözüm uzayının küçük olmasını bu da aranan en iyi çözüm değerlerine ulaşılamamasına neden olmaktadır. Çözüm uzayının çok büyük değerde olması ise hem genetik algoritmanın etkinliğini azaltmakta hem de çözümün farklı noktalarda aranmasına neden olmaktadır .

Gen, Kromozom ve Populasyon Yapısı



3.4 Kodlama

İlk adım; problem için arama uzayını en iyi temsil eden kodlama yapısının seçilmiş olmasıdır. Kodlama biçimi, genetik algorimanın performansını oldukça önemli oranda etkiler; fakat programa bağlı olduğundan bütün problemler için geçerli en uygun kodlama biçimini söylemek imkânsızdır

3.4.1 İkili (Binary) Kodlama

En yaygın olarak kullanılan kodlama türüdür. Burada kromozomlar 0 ve 1 şeklinde gen değerlerinde kodlanırlar. Bu dizideki her bit, çözümün belli karakteristiğini temsil eder veya tüm dizi bir sayıyı temsil eder.

İkili Düzende Kodlama Yapısı

Kromozom 1	1 0 1 0 1 1 0 1 1
Kromozom 2	0 0 1 0 1 0 1 0 1

3.4.2 Sıralı (Permütasyon) Kodlama

Sıralı (permütasyon) kodlama tekniği genellikle gezgin satıcı, çizelgeleme ve sıralama, şebeke tasarımları gibi sıra takibi olan problemlere uygulanır.

	1	2	3	4	5	6	7	8	9	10	11	12
A_1:	1	2	3	4	5	6	7	8	9	0	4	5
A_2:	7	4	6	1	2	8	3	5	3	1	9	6
Kalıp	1	0	0	0	1	0	1	0	0	0	0	0
Çaprazlama Sonrası Oluşan Yeni Bireyler												
A^1_1:	1	7	2	4	5	6	3	8	9	0	4	5
A^1_2:	1	4	6	5	2	8	3	7	3	1	9	6

3.4.3 Değer (Alpha-Numeric Encoding) Kodlaması

Değer kodlama yönteminde ilgili parametre değerleri doğrudan alınır. Özel problemlerde alfa-sayısal ya da gerçel sayılar olarak kullanımı gerçekleştirilir

Değer kodlama yapısı

Kromozom A	0.123 1.234 2.345 4.567 5.678
Kromozom B	A B C D E F G H I J K L M N

3.5 Seçim

Goldberg'e göre, yeni nesiller için ebeveyn kromozomların belirlenmesi sürecinde önceki populasyondan gelen bazı kromozomların yeni populasyona

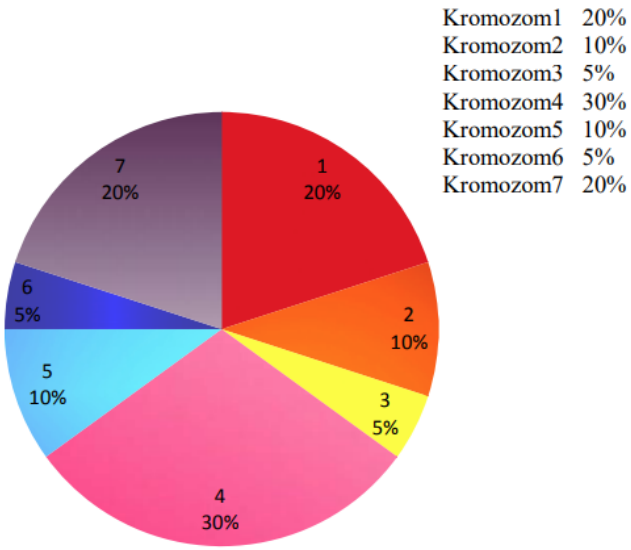
aktırılması gerekmektedir. Burada önemli olan bu kromozomların nasıl seçileceğidir. Darvin evrim teorisine göre; hayatta kalan en iyi kromozom ebeveyn olarak yeni oğul kromozomları oluşturur. En iyi kromozom seçilmesinde birçok yöntem bulunmaktadır. Her bir kromozomun uygunluk değeri hesaplandıktan sonra uygunluğu yüksek olan kromozomların seçilmesi için geliştirilmiş değişik seçim yöntemleri bulunmaktadır

3.5.1 Rulet Tekeri Seçim Yöntemi

Genetik algoritmalarda kullanılan en basit ve en yaygın seçim mekanizması rulet tekerleği (çemberi) seçimidir.

Rulet Tekeri seçim operatöründe, bütün kromozomlar uygunluk değerlerine göre bir rulet etrafında dizilirler. Rulet üzerinde uygunluk değerlerine göre sıralanan kromozomlar rasgele olarak seçilirler. Bu şekilde her birey seçilmek için kendi uygunluk değerine göre bu rulet tekerinden bir pay almaktadır. Daha büyük alana sahip bireyin seçilme şansı daha fazla olacaktır. Bu metot yardımıyla kromozomlar istatistiksel yöntemler kullanılarak uygunluk fonksiyonu değerlerinin toplam uygunluk fonksiyonuna oranları ölçüsünde seçilirler. Ancak bu seçim yöntemi, uyum değeri büyük olan bireylerin seçilme olasılığı yüksek olduğu için hep aynı kromozomların seçilmesine neden olmaktadır. Bu da populasyon içindeki çeşitliliği etkileyerek sorun yaratır.

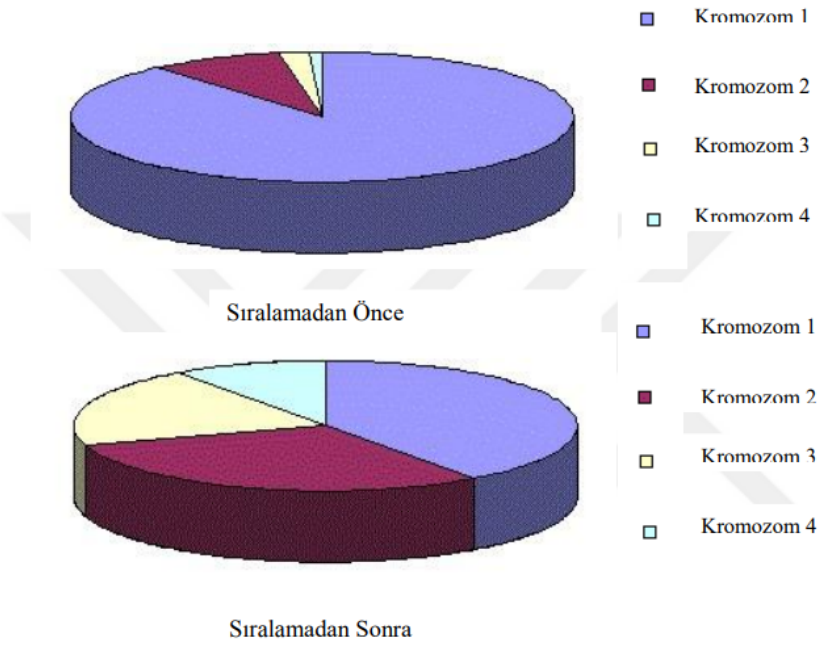
Rulet Tekerleği Seçme Operatörü



3.5.2 Sıralı (Rank) Seçim Yöntemi

Rulet seçimi eğer uyumluluk çok fazla değişiyorsa sorun çıkartabilir. Örneğin en iyi kromozomun uyumluluğu %90 ise diğer kromozomların seçilme şansı azalacaktır. Bunu önlemek için sıralı seçim kullanılabilir. Sıralı seçimde en kötü uyumlulukta olan kromozoma 1 değeri sonrakine 2 değeri verilir ve böylelikle seçilmede bunlara öncelik tanınmış olur. Bu şekilde onların da seçilme şansı artar. Fakat bu da çözümün daha geç yakınsamasına neden olabilir[11]

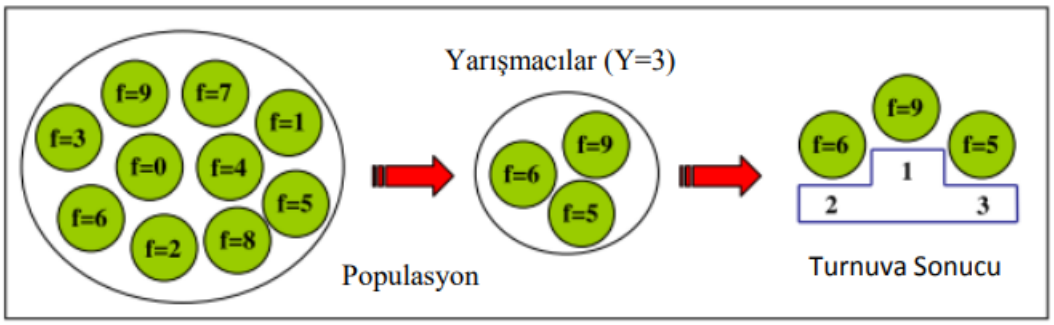
Sıralı Seçim



3.5.3 Turnuva Seçim Yöntemi

Bu seçim yönteminde, bireyler rasgele olarak gruplanır ve gruptaki bireyler aralarında seçim işlemi yapılmak üzere rekabete sokulur. Grup içinde en yüksek uygunluk değerine sahip olan birey, yeni nesli oluşturmak için ebeveyn bireylerden biri olarak seçilir. Bu işlem, toplam birey sayısına ulaşıncaya kadar devam eder. Bazı uygulamalarda grup büyüklüğü iki olarak seçilirken, bazılarında çok daha büyük gruplar oluşturulur [12]

Turnuva Seçim Yöntemi



Populasyondaki bireylerin uygunluk değerleri f ile bireyler arasından rasgele seçilen grup büyüklüğü Y ile gösterilmiştir

3.5.4 Sabit Durum (Kararlı Hal) Seçim Yöntemi

Sabit durum metodunda, her nesilde yalnızca birkaç birey yer değiştirir. Çoğunlukla çok düşük uygunluk değerine sahip bireyler, çaprazlama ve mutasyon yöntemleriyle yeniden üretilerek yeni nesilde yer alırlar. Sabit durumlu GA'lar daha çok kural tabanlı sistemlerde kullanılır

3.5.5 Seçkinlik (Elitizm) İşlemi

Mansfield'e göre üreme, çaprazlama ve mutasyon işlemleri sonrasında kuşakta bulunan en iyi uygunluk değerine sahip birey, sonraki kuşağa aktarılamayabilir. Bunu önlemek için bu işlemlerden sonra oluşan yeni kuşağa bir önceki kuşağın en iyi (elit) bireyi, yeni kuşaktaki herhangi bir birey ile değiştirilir.

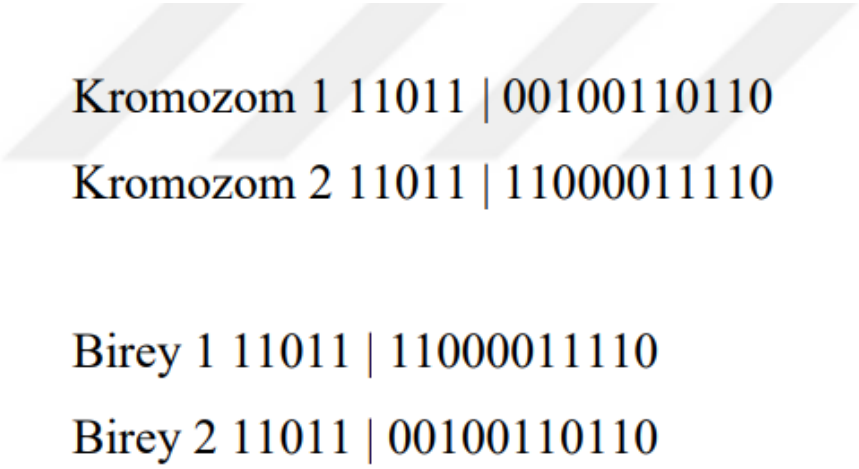
3.6 Genetik Operatörler

Genetik algoritmalarda belirli noktalardan sonra nesil çeşitliliği olmamaktadır. Bunun için dizilere çaprazlama (crossing over) ve değişim (mutation) operatörleri belirli yüzdelik oranlarıyla uygulanarak nesil çeşitliliği sağlanır. Böylece çözümün belirli noktalara gelip tıkanması önlenmiş olur

3.6.1 Çaprazlama

[13]

Çaprazlama ebeveynlerden bazı genleri alarak yeni bireyler oluşturma işlemidir. Burada amaç, eldeki nesilden farklı nesiller elde etmektir. Çaprazlama yapılacak konum rastgele seçilir. Oluşan yeni birey ebeveynlerin bazı özelliklerini almış ve bir bakıma ikisinin kopyası olmuştur. Daha iyi performans almak amacıyla değişik çaprazlamalar kullanılabilir



Çaprazlama operatörü, iki dizinin bir araya gelerek karşılıklı gen yapılarının değişimi ile yeni dizilerin oluşumunu sağlayan operatördür. Çaprazlanarak gen değişiminin yapılmasından önce dizilerin çaprazlamaya tutulma olasılığı belirlenmelidir. Literatürde bu oranın %50-%95 oranında uygulandığı görülmektedir.

Çaprazlamada bir diğer önemli unsur ise ne tür bir çaprazlamanın yapılacağıdır.

3.6.2 İkili kodlama Düzeninde Çaprazlama Yöntemleri

İkili kodlama düzeni için çaprazlama yöntemleri tek nokta, iki nokta ve üç nokta çaprazlama yöntemleri olarak sınıflandırılmıştı.

- **Tek Nokta Çaprazlama Operatörü:**
  Bu işlemde kromozomlar rastgele bir yerinden kesilir ve sonra ilgili genler ile yer değiştirilir. [13]

Tek Noktalı Çaprazlama

	1	2	3	4	5	6	7	8	9	10	11	12
A_1:	1	0	0	1	1	1	0	0	1	1	0	1
A_2:	1	1	0	0	0	0	1	1	1	0	0	0
Çaprazlama Sonrası Oluşan Yeni Bireyler												
A^1_1:	1	0	0	1	1	1	0	0	1	0	0	0
A^1_2:	1	1	0	0	0	0	1	1	1	1	0	1

- **İki Nokta Çaprazlama Operatörü:**
  Bazı durumlarda tek noktalı çaprazlama yöntemi yetersiz kalabilir ya da büyük parçalı blokların bozulması performansı düşürebilir. Bu sebeple iki noktalı çaprazlama yöntemi tercih edilebilir.İki noktalı çaprazlama operatöründe, rasgele iki nokta seçilir ve bu iki nokta arasında kalan bloklar kromozomlar arasında yer değiştirilir. Bu yöntem populasyondaki kromozomların performansını arttırabilir.[12]

İki Noktalı Çaprazlama

	1	2	3	4	5	6	7	8	9	10	11	12
A_1:	1	0	0	1	1	1	0	0	1	1	0	1
A_2:	1	1	0	0	0	0	1	1	1	0	0	0
Çaprazlama Sonrası Oluşan Yeni Bireyler												
A^1_1:	1	0	0	1	1	0	0	1	1	1	0	1
A^1_2:	1	1	0	0	1	1	0	0	1	0	0	0

3.6.3 Sıralı Kodlama Düzeninde Çaprazlama Yöntemleri

Çizelgeleme problemlerinde sıkça kullanılan sıralı (permütasyon) kodlama düzeninde yer alan çeşitli çaprazlama yöntemleri mevcuttur

- **Pozisyona dayalı çaprazlama operatörü (PBX):** bu yöntemde çaprazlama kalıp olarak sabit kalacak olan gen hücrelerini belirler. Kalıpla işaretlenen noktalar dizide sabit kalırken diğer noktalarda iki birey yer değiştirilerek yeni bireylerin üremesi sağlanır.[14]

Pozisyona Dayalı Çaprazlama

	1	2	3	4	5	6	7	8	9	10	11	12
A_1:	3	4	7	1	1	0	4	8	9	2	3	3
A_2:	0	0	1	4	7	2	8	9	2	1	0	0
Kalıp	1	1	1	0	0	0	1	1	0	0	1	0
Çaprazlama Sonrası Oluşan Yeni Bireyler												
A^1_1:	3	4	7	4	7	2	4	8	2	1	3	0
A^1_2:	0	0	1	1	1	0	8	9	9	2	0	3

- **Sıraya dayalı çaprazlama operatörü (OBX):** kalıp üzerindeki 1 değerleri çaprazlamada kullanılacak değerleri gösterir. Bu tür çaprazlama, kromozomu oluşturan karakterlerin sayı ve sıralarının önem taşıdığı durumlarda kullanılır.

Sıraya Dayalı Çaprazlama

	1	2	3	4	5	6	7	8	9	10	11	12
A_1:	1	2	3	4	5	6	7	8	9	0	4	5
A_2:	7	4	6	1	2	8	3	5	3	1	9	6
Kalıp	1	0	0	0	1	0	1	0	0	0	0	0
Çaprazlama Sonrası Oluşan Yeni Bireyler												
A^1_1:	1	7	2	4	5	6	3	8	9	0	4	5
A^1_2:	1	4	6	5	2	8	3	7	3	1	9	6

- **Kısmi eşleşmeli çaprazlama operatörü (PMX):**iki bireyden rastgele bir aralık belirlenir. Bu aralıktaki değerler yer değiştirilir

Kısmi Planlı Çaprazlama

	1	2	3	4	5	6	7	8
A_1:	2	8	6	4	5	7	1	3
A_2:	8	7	2	1	3	4	5	6
Çaprazlama								
A^1_1:	2	8	2	1	3	7	1	3
A^1_2:	8	7	6	4	5	4	5	6

Yer değiştirme sonunda dizide aynı olan değerler değiştirilen değerlerle tamamlanır.

	1	2	3	4	5	6	7	8
Oluşan Yeni Bireyler								
A^1_1:	6	8	2	1	3	7	4	5
A^1_2:	8	7	6	4	5	1	2	3

3.7 Mutasyon

[2]
Kromozomların genleri veya genleri oluşturan küçük birimleri üzerinde değişiklik yapılmasını sağlayan işlemcidir. Değişime uğratılacak kromozomun seçiminde, kromozomun değişime uğrama ihtimalini gösteren ve başlangıçta sabit olarak tanımlanan bir değişim oranı söz konusudur. Genetik algoritmalarda değişime tabi tutulacak kromozomların belirlenmesinde bazılarının istisna tutulması veya özellikle değişime uğratılması gibi özel stratejiler tanımlanabilir.

Amaç belli bir nesil sayısından sonra populasyon içerisindeki bireylerin gitgide birbirlerine benzemesine engel olmaktır. Çünkü bu durum çözüm uzayının daralmasına neden olmaktadır. Bireylere ne kadar çaprazlama operatörü uygulansa da belli bir nesil sayısından sonra birey çeşitliliği sağlanmamaktadır. Bu durumda bireyi oluşturan genlerden rasgele bir tanesi seçilir. Rasgele seçilen genin değeri değiştirilir. Böylelikle populasyon içindeki bireylerin çeşitliliğinin devamı sağlanmış olunur . Yapay sistemlerde mutasyon işlemi esnasında kromozomdaki gen sayısı değişmez sabit kalır. Doğal populasyondaki mutasyon oranı oldukça düşüktür. Mutasyon frekansının büyüklüğü GA'nın performansını etkilemektedir. Mutasyon işlemi bir tek kromozom üzerinde yapılır

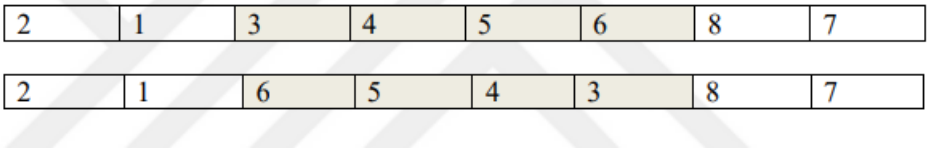
Mutasyon oranı, tıpkı çaprazlama oranında olduğu gibi mutasyonun olasılığını ifade eden bir orandır. Yine rasgele yöntemlerle kromozomun mutasyona uğrayıp uğramayacağı belirlenir ve buna göre mutasyon gerçekleştirilir. Mutasyon oranı genellikle çok düşük (0,01) olduğundan mutasyon işlemi fertlerde az görülür .

Mutasyonun sağladığı avantaj, problemin çözüm alanını araştırmada yön değişikliklerini sağlayarak araştırmanın kısır döngüye girmesini önlemektir. Mutasyon yöntemleri genel olarak beş farklı şekilde sınıflandırılmıştır

3.7.1 Ters Mutasyon

Bir bireyde rassal olarak iki pozisyon seçilir, bu iki pozisyondaki alt diziler ters çevrilir

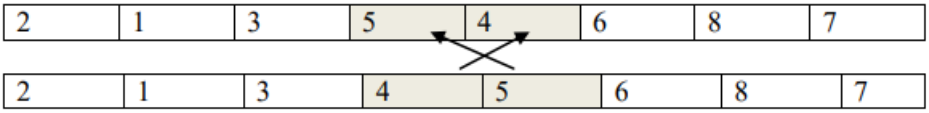
Ters Mutasyon



3.7.2 Komşu İki Geni Değiştirme

Rassal olarak iki komşu iş yer değiştirebilir.

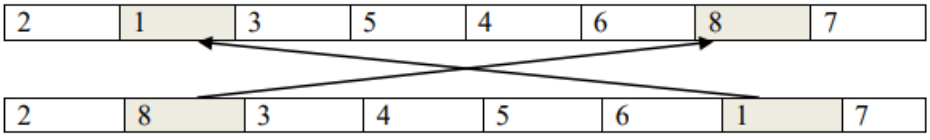
Komşu İki Geni Değiştirme



3.7.3 Keyfi İki Gen Değiştirme

Rassal olarak seçilen iki iş değiştirilebilir. Özel bir durum olarak, değiştirilebilen iki komşu işi bu mutasyon içerir

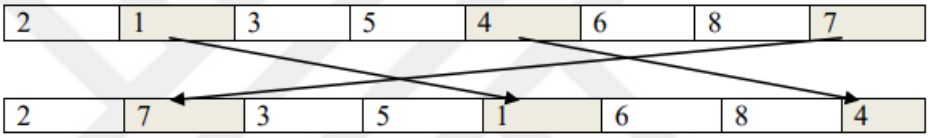
Keyfi İki Gen Değiştirme



3.7.4 Keyfi Üç Gen Değiştirme

Rassal seçilen üç iş keyfi olarak değiştirilir

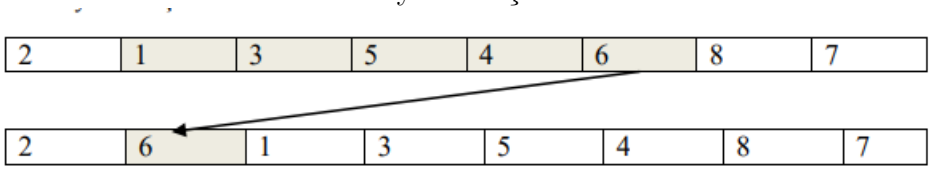
Keyfi Üç Gen Değiştirme



3.7.5 Araya Yerleştirme

Rassal olarak seçilen bir kaydırma noktasında kromozomdaki bir iş kaydırılır ve diğer bir pozisyona yerleştirilir. Komşu iki iş değiştirme yönteminin özel bir durumudur. Keyfi üç iş değiştirmeyle bir kesişime sahiptir

Araya Yerleştirme



3.8 Durdurma Kriteri

Üreme, çaprazlama ve mutasyon işlemlerinden sonra yeni bir nesil oluşmaktadır. Tüm bu işlemler sonsuz döngü içerisinde yapılır. Eğer bir durdurma kriteri belirlenmez ise bu süreç sonsuza kadar devam eder.

A. **Hesaplama zamanı kriteri:**
Bu yöntemde önceden bir hesaplama zamanı veya döngü sayısı belirlenmekte, bu zaman veya döngü sayısına ulaşıldığında durdurulmaktadır. Bu yöntemde belirlenen döngü sayısı gerektiğinden fazla ya da eksik olabilir.

B. **Optimizasyon hedefi kriteri:**
Önceden ulaşılması istenen amaç fonksiyonu değeri bilinmektedir. Uyum değeri bu değere ulaştığında algoritma durdurulmaktadır.

C. **Minimum iyileşme kriteri:**
Genetik algoritma problemlerinde bulunan en iyi çözümler önce hızlı daha sonra yavaş yavaş artış göstermektedir. Bulunan değerlerdeki iyileşme hızının giderek azalması ve sıfıra yaklaşması, artık daha fazla

iyileşme beklenmemesi gerektiğini gösterebilir. Çözüme harcanacak zaman ile çözümden beklenecek kalite arasında bir denge kurularak durdurma gerçekleştirilir[13]

3.9 Genetik Algoritmaların Çözümünde Takip Edilecek İşlem Adımları

1. Adım 1: Kullanıcının önceden tanımladığı kurallara göre genellikle rassal bir çözüm grubu seçilir veya kullanıcının kendisi ilk çözüm grubunu belirleyebilir. İlk çözüm grubuna başlangıç populasyonu denir.

2. Adım 2: Her bir kromozom için bir uygunluk değeri hesaplanır; bulunan uygunluk değerleri dizilerin çözüm kalitesini gösterir. Populasyonda yer alan en iyi uygunluk değerine sahip olan birey (kromozom), bir sonraki yeni nesile (populasyon) doğrudan değiştirilmeden aktarılır.

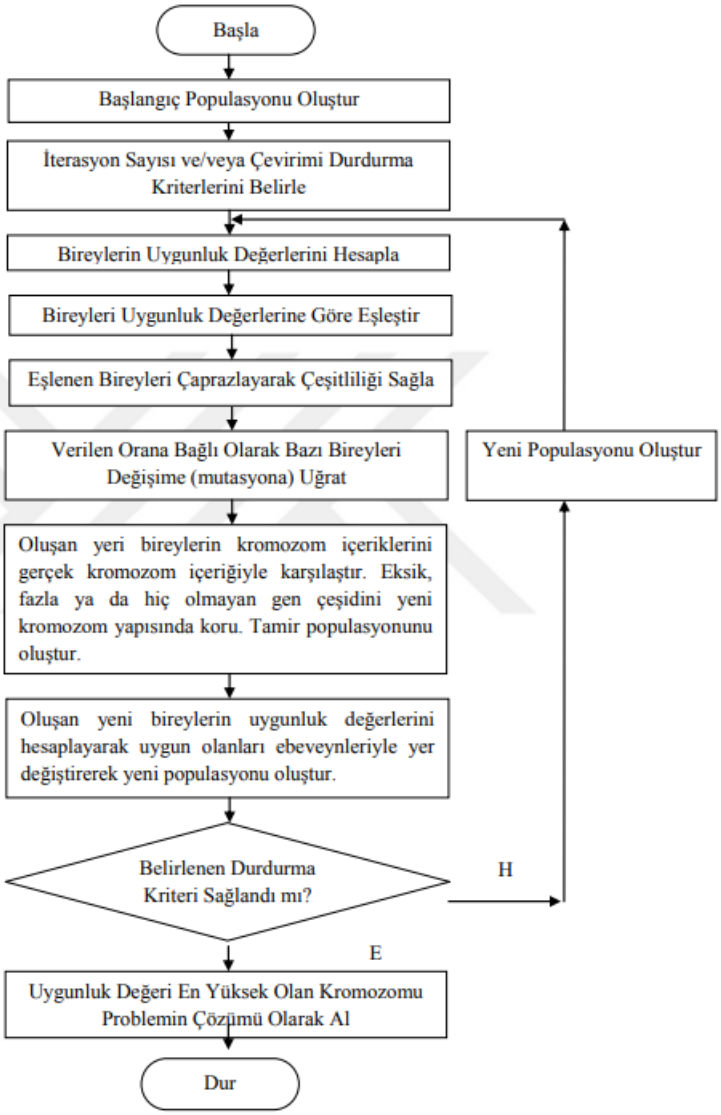
3. Adım 3: İki grup dizi (kromozom), belirli bir seçim yöntemine (uygunluk değerlerine) göre rassal olarak seçilir.

4. Adım 4: Seçilen iki kromozom için rassal olarak genetik operatörler kullanılarak çaprazlama işlemi gerçekleştirilir. Sonuçta yeni populasyonda yer alacak iki yeni birey (kromozom) oluşur. Çaprazlama, yeni populasyonda yer alacak birey sayısına ulaşılana dek sürer.

5. Adım 5: Yeni populasyondaki bireyler, rassal olarak mutasyon işleminden geçerler.

6. Adım 6: Önceden belirlenen nesil sayısı boyunca yukarıdaki işlemler sürdürülür. Eğer en büyük nesil sayısına ulaşılmamışsa Adım 2'ye dönülür. En büyük nesil sayısına ulaşınca işlem bitirilir. Uygunluk değeri en yüksek olan kromozom (çözüm) seçilir.

Genetik Algoritmanın Akış Diyagramı [13]



3.10 Yapılan Çalışmalar

Bu bölümden yukarıda anlatılan Genetik Algoritma kavramlarını anlaşılması icin basit uygulamalar yapılmıştır.

3.10.1 Örnek-1

[15]
Uygulamada ki amac belirli bir bit dizisi (hedef gen) oluşturmaktır. Başlangıçta rastgele bir popülasyon oluşturulur ve ardından iteratif olarak en uygun genlere yaklaşmak için çaprazlama ve mutasyon işlemleri gerçekleştirilir.
Uygunluk fonksiyonu, hedef gene ne kadar yakın olduklarını belirler.

1. **İlk olarak, random modülünü ve genetik algoritmanın parametrelerini tanımlıyoruz:**

```
1   import random
2
3   # Genetik algoritmanın parametreleri
4   population_size = 10
5   gene_length = 8
6   mutation_rate = 0.1
7
8   # Hedef gen
9   target_gene = "10101010"
10
```

- Popülasyon boyutu, yani her bir neslin kaç bireyden oluşacağı.
- Her bir genin uzunluğu. Bu örnekte genler bit dizileri olarak temsil ediliyor.
- Mutasyon oranı, yani her bir genin mutasyona uğrama olasılığı.
- Hedef gen, yani genetik algoritmanın hedeflediği bit dizisi.

2. **'create population'**
   Fonksiyonu, başlangıç popülasyonunu oluşturur. Rastgele bit dizileri oluşturarak popülasyonu doldurur.

```
10
11  # Popülasyon oluşturma
12  def create_population():
13      population = []
14      for _ in range(population_size):
15          gene = ''.join(random.choice("01") for _ in range(gene_length))
16          population.append(gene)
17      return population
18
```

3. **'fitness'**
   Fonksiyonu, bir genin uygunluk değerini hesaplar. Uygunluk değeri, hedef gene ne kadar yakın olduğunu belirten bir skordur.

```
18
19  # Uygunluk değerini hesaplama
20  def fitness(gene):
21      score = 0
22      for i in range(len(gene)):
23          if gene[i] == target_gene[i]:
24              score += 1
25      return score
26
```

4. **rank population fonksiyonu**
   Popülasyonu uygunluk değerine göre sıralar. En yüksek uygunluk değerine sahip genler en üstte olacak şekilde sıralanır.

```
26
27   # Popülasyonu sıralama
28   def rank_population(population):
29       return sorted(population,key=lambda gene:fitness(gene),reverse=True)
30
```

5. **select parents fonksiyonu**
   Ebeveyn seçimi yapar. Bu örnekte turnuva seçimi kullanılmıştır; rastgele iki birey seçilir ve onlardan uygunluk değeri daha yüksek olanı ebeveyn olarak seçilir

```
31   # Ebeveyn seçimi (turnuva seçimi)
32   def select_parents(population):
33       parent1 = random.choice(population)
34       parent2 = random.choice(population)
35       return parent1, parent2
36
```

6. **crossover fonksiyonu**
   Çaprazlama işlemini gerçekleştirir. İki ebeveynin genlerini alır ve belirli bir noktadan çaprazlar, böylece iki yeni çocuk gen oluşturur.

```
# Çaprazlama (iki nokta çaprazlama)
def crossover(parent1, parent2):
    crossover_point1 = random.randint(0, gene_length - 1)
    crossover_point2 = random.randint(crossover_point1, gene_length)
    child1 = parent1[:crossover_point1] + parent2[crossover_point1:crossover_point2] + parent1[crossover_point2:]
    child2 = parent2[:crossover_point1] + parent1[crossover_point1:crossover_point2] + parent2[crossover_point2:]
    return child1, child2
```

7. **mutate fonksiyonu**
   Genlerde mutasyon gerçekleştirir. Her bir bitin mutasyona uğrama olasılığı mutation rate parametresi tarafından belirlenir.

```
44
45  # Mutasyon
46  def mutate(gene):
47      mutated_gene = ''
48      for bit in gene:
49          if random.random() < mutation_rate:
50              mutated_gene += '1' if bit == '0' else '0'
51          else:
52              mutated_gene += bit
53      return mutated_gene
54
```

8. **create new generation fonksiyonu**
   Bir sonraki nesli oluşturur. Ebeveynleri seçer, çaprazlama ve mutasyon işlemlerini uygular, ve yeni nesli oluşturur.

```
54
55  # Yeni nesil oluşturma
56  def create_new_generation(population):
57      new_generation = []
58      while len(new_generation) < population_size:
59          parent1, parent2 = select_parents(population)
60          child1, child2 = crossover(parent1, parent2)
61          child1 = mutate(child1)
62          child2 = mutate(child2)
63          new_generation.extend([child1, child2])
64      return new_generation[:population_size]
```

9. **genetic algorithm fonksiyonu**
   Genetik algoritmayı çalıştırır. Başlangıç popülasyonu oluşturur, her bir nesilde en iyi geni ve uygunluk değerini gösterir, ve hedef gene ulaşılana kadar yeni nesiller oluşturur.

```
65
66  # Genetik algoritma
67  def genetic_algorithm():
68      population = create_population()
69      generation = 1
70      while True:
71          population = rank_population(population)
72          best_gene = population[0]
73          print(f"Generation {generation}, Best gene: {best_gene}, Fitness: {fitness(best_gene)}")
74          if best_gene == target_gene:
75              print("Target gene found!")
76              break
77          population = create_new_generation(population)
78          generation += 1
79
```

10. **main**
    Genetic algorithm fonksiyonu çağrılır ve genetik algoritma çalıştırılır.

```
80  if __name__ == "__main__":
81      genetic_algorithm()
82
```

Bu adımların birleşimi, genetik algoritmanın çalışmasını sağlar. Başlangıçta rastgele bir popülasyon oluşturulur, her bir nesilde en iyi genler seçilir ve çaprazlama ve mutasyon işlemleri uygulanarak yeni nesiller oluşturulur. Bu süreç, hedef gene ulaşılana kadar devam eder.

```
Generation 1, Best gene: 10001100, Fitness: 5
Generation 2, Best gene: 00001010, Fitness: 6
Generation 3, Best gene: 10011000, Fitness: 5
Generation 4, Best gene: 10000111, Fitness: 4
Generation 5, Best gene: 10011000, Fitness: 5
Generation 6, Best gene: 10001001, Fitness: 5
Generation 7, Best gene: 10001010, Fitness: 7
Generation 8, Best gene: 11101010, Fitness: 7
Generation 9, Best gene: 11101000, Fitness: 6
Generation 10, Best gene: 11001000, Fitness: 5
Generation 11, Best gene: 11100000, Fitness: 5
Generation 12, Best gene: 11100000, Fitness: 5
Generation 13, Best gene: 10000010, Fitness: 6
Generation 14, Best gene: 10111011, Fitness: 6
Generation 15, Best gene: 10111011, Fitness: 6
Generation 16, Best gene: 10001111, Fitness: 5
Generation 17, Best gene: 10001110, Fitness: 6
Generation 18, Best gene: 10101100, Fitness: 6
Generation 19, Best gene: 00111011, Fitness: 5
Generation 20, Best gene: 01111010, Fitness: 5
Generation 21, Best gene: 11101001, Fitness: 5
Generation 22, Best gene: 10110010, Fitness: 6
Generation 23, Best gene: 10111010, Fitness: 7
Generation 24, Best gene: 10101010, Fitness: 8
Target gene found!
```

Bu çıktı, genetik algoritmanın her bir nesilindeki en iyi geni ve bu genin uygunluk değerini gösteriyor. Her bir satır, bir jenerasyonu temsil eder.

Örneğin, "Generation 1, Best gene: 10001100, Fitness: 5" satırı ilk jenerasyonun en iyi genini (10001100) ve bu genin uygunluk değerini (Fitness: 5) gösterir. Daha sonra, her jenerasyonda en iyi gen ve uygunluk değeri güncellenir.

Son jenerasyonda (Generation 24), hedef gen olan "10101010" elde edilmiştir ve uygunluk değeri 8'dir. Bu, genetik algoritmanın hedef gende yaklaştığını ve sonunda hedefi başarıyla bulduğunu gösterir.

3.11 Örnek-2

Bu bölümün ikinci çalışmasında Genetik Algoritma kullanılarak 'örnek-1'adlı çalışmadan esinlenerek yapılmıştır.Uygulamada ki amac hedef sayısı olan 1905'ı bulmaktır.

```python
import random
# Parametreler
populasyon_boyutu = 100
mutasyon_orani = 0.1
nesil_sayisi = 1000
# Hedef sayı
hedef_sayi = 1905

# Başlangıç popülasyonu oluşturma
def birey_olustur(uzunluk):
    return [random.randint(0, 9) for _ in range(uzunluk)]

# Uygunluk (fitness) hesaplama
def uygunluk_hesapla(birey):
    uygunluk = 0
    for i in range(len(birey)):
        uygunluk += abs(birey[i] - int(str(hedef_sayi)[i]))
    return uygunluk

# Çaprazlama
def caprazlama(ebeveyn1, ebeveyn2):
    caprazlama_noktasi = random.randint(1, len(ebeveyn1) - 1)
    cocuk1 = ebeveyn1[:caprazlama_noktasi] + ebeveyn2[caprazlama_noktasi:]
    cocuk2 = ebeveyn2[:caprazlama_noktasi] + ebeveyn1[caprazlama_noktasi:]
    return cocuk1, cocuk2

# Mutasyon
def mutasyon(birey, mutasyon_orani):
    for i in range(len(birey)):
        if random.random() < mutasyon_orani:
            birey[i] = random.randint(0, 9)
    return birey

```

```python

# Genetik algoritma
def genetik_algoritma(populasyon_boyutu, mutasyon_orani, nesil_sayisi):
    populasyon = [birey_olustur(len(str(hedef_sayi))) for _ in range(populasyon_boyutu)]

    for nesil in range(nesil_sayisi):
        populasyon = sorted(populasyon, key=lambda x: uygunluk_hesapla(x))
        en_iyi_birey = populasyon[0]
        print(f"Nesil {nesil + 1} - En İyi Birey: {''.join(map(str, en_iyi_birey))}, Uygunluk: {uygunluk_hesapla(en_iyi_birey)}")

        if uygunluk_hesapla(en_iyi_birey) == 0:
            print("Hedef sayı bulundu:", ''.join(map(str, en_iyi_birey)))
            return en_iyi_birey

        yeni_nesil = []

        for _ in range(populasyon_boyutu // 2):
            ebeveyn1 = random.choice(populasyon)
            ebeveyn2 = random.choice(populasyon)
            cocuk1, cocuk2 = caprazlama(ebeveyn1, ebeveyn2)
            cocuk1 = mutasyon(cocuk1, mutasyon_orani)
            cocuk2 = mutasyon(cocuk2, mutasyon_orani)
            yeni_nesil.extend([cocuk1, cocuk2])

        populasyon = yeni_nesil

    print("Belirli nesil sayısına ulaşıldı ancak hedef sayı bulunamadı.")
    en_iyi_birey = min(populasyon, key=lambda x: uygunluk_hesapla(x))
    print("En yakın sayı:", ''.join(map(str, en_iyi_birey)))
    return en_iyi_birey


# Genetik algoritma çalıştırma
genetik_algoritma(populasyon_boyutu, mutasyon_orani, nesil_sayisi)

```

- Parametreler: populasyon boyutu, mutasyon orani ve nesil sayisi gibi parametreler, genetik algoritmanın çalışma şeklini belirler.
- Hedef Sayı Belirleme: İlk olarak, hedef sayıyı 1905 olarak belirliyoruz.
- Başlangıç Popülasyonu Oluşturma: birey olustur fonksiyonu, belirli bir uzunluktaki bireyi rastgele oluşturur. Her bir birey, hedef sayıdaki rakamların sayısına sahip olacak şekilde 0 ile 9 arasında rastgele rakamlardan oluşur.
- Uygunluk (Fitness) Hesaplama: uygunluk hesapla fonksiyonu, bir bireyin hedef sayıya ne kadar uygun olduğunu hesaplar. Her bir rakamın hedef sayıdaki rakamla ne kadar uyumlu olduğunu belirlemek için mutlak değer farkları toplanır.

- Çaprazlama: caprazlama fonksiyonu, iki ebeveyn bireyden yeni çocuk bireyler üretir. Rastgele bir çaprazlama noktası belirlenir ve bu noktaya kadar olan kısımlar ebeveynler arasında değiştirilerek çocuklar oluşturulur.

- Mutasyon: mutasyon fonksiyonu, bir bireyin genlerinde belirli bir mutasyon oranıyla rastgele değişiklikler yapar. Her bir gen için belirli bir olasılıkla (mutasyon oranına bağlı olarak) yeni bir rastgele rakam atanır.

- Genetik Algoritma: genetik algoritma fonksiyonu, genetik algoritmayı uygular. Belirli bir populasyon boyutu ve nesil sayısı ile başlar. Her bir nesilde, populasyon uygunluklarına göre sıralanır ve en iyi birey ekrana yazdırılır. Eğer hedef sayı bulunursa veya belirli bir nesil sayısına ulaşılırsa döngüden çıkılır.

- Genetik Algoritma Çalıştırma: En son kısımda ise parametrelerle genetik algoritma çalıştırılır.

```
Nesil 1 - En İyi Birey: 1715, Uygunluk: 3
Nesil 2 - En İyi Birey: 0906, Uygunluk: 2
Nesil 3 - En İyi Birey: 3902, Uygunluk: 5
Nesil 4 - En İyi Birey: 1807, Uygunluk: 3
Nesil 5 - En İyi Birey: 1835, Uygunluk: 4
Nesil 6 - En İyi Birey: 1802, Uygunluk: 4
Nesil 7 - En İyi Birey: 3904, Uygunluk: 3
Nesil 8 - En İyi Birey: 0605, Uygunluk: 4
Nesil 9 - En İyi Birey: 1704, Uygunluk: 3
Nesil 10 - En İyi Birey: 1404, Uygunluk: 6
Nesil 11 - En İyi Birey: 1704, Uygunluk: 3
Nesil 12 - En İyi Birey: 0704, Uygunluk: 4
Nesil 13 - En İyi Birey: 1816, Uygunluk: 3
Nesil 14 - En İyi Birey: 1504, Uygunluk: 5
Nesil 15 - En İyi Birey: 0802, Uygunluk: 5
Nesil 16 - En İyi Birey: 2908, Uygunluk: 4
Nesil 17 - En İyi Birey: 1904, Uygunluk: 1
Nesil 18 - En İyi Birey: 0802, Uygunluk: 5
Nesil 19 - En İyi Birey: 2906, Uygunluk: 2
Nesil 20 - En İyi Birey: 2904, Uygunluk: 2
Nesil 21 - En İyi Birey: 1706, Uygunluk: 3
Nesil 22 - En İyi Birey: 2803, Uygunluk: 4
Nesil 23 - En İyi Birey: 2804, Uygunluk: 3
Nesil 24 - En İyi Birey: 1814, Uygunluk: 3
Nesil 25 - En İyi Birey: 3814, Uygunluk: 5
Nesil 26 - En İyi Birey: 2706, Uygunluk: 4
Nesil 27 - En İyi Birey: 1906, Uygunluk: 1
Nesil 28 - En İyi Birey: 1905, Uygunluk: 0
Hedef sayı bulundu: 1905
```

Bu çıktı, genetik algoritmanın her neslindeki en iyi bireyi ve uygunluk değerini göstermektedir.
Örneğin, "Nesil 1 - En İyi Birey: 1715, Uygunluk: 3" ifadesi, ilk neslin en iyi bireyinin 1715 olduğunu ve bu bireyin uygunluk değerinin 3 olduğunu belirtir. Uygunluk değeri, hedef sayıya ne kadar yakın olduğunu gösterir. Daha küçük bir uygunluk değeri, hedef sayıya daha yakın bir çözümü temsil eder.
Çıktıda görüldüğü gibi, 28. nesilde uygunluk değeri 0 olan bir birey bulunmuştur. Bu, hedef sayının tam olarak bulunduğunu gösterir. Sonuç olarak, "Hedef sayı bulundu: 1905" ifadesiyle belirtilir.

3.12 Çizelgeleme ve Genetik Algoritmalar

Genetik algoritmaların çizelgeleme problemlerinde kullanımı, ilk olarak 1985 yılında Davis tarafından yapılmıştır. Davis'in çalışmasında, bir iş atölyesinde kârı maksimize etmek için çizelgelemenin önemi vurgulanmıştır. Bu çalışma, gerçekçi olmaktan ziyade basit ve öğretici bir çizelgeleme problemi üzerinde genetik algoritmaların nasıl uygulanabileceğini göstermiştir. Davis'in bu çalışması, sonraki araştırmalar için önemli bir temel oluşturmuştur.

Genetik algoritmalar çizelgeleme problemlerinin çözümünde önemli bir rol oynamakta ve bu alanda yapılan çalışmalar, bu algoritmaların etkinliğini ve uygulanabilirliğini göstermektedir. Gelecekteki çalışmalar, genetik algoritmaların çizelgeleme problemlerindeki performansını artırmak ve daha karmaşık problemleri ele almak için yöntemlerin geliştirilmesine sağlayacaktır.

Önceki bölümde genetik algoritmalar detaylı bir şekilde incelenmiş ve yapısı ile kullanım detayları üzerinde durulmuştu. Bu bölümde ise genetik algoritmaların çizelgeleme problemlerinde nasıl kullanıldığı ele alınmış ve örnek çalışmalar özetlenmiştir.

3.12.1 Örnek-1

Bu örneğimizde Genetik Algoritma kullanılarak cizelgeleme problemleri ele alınmıştır. Genetik Algoritma kullanılarak bir Ders programı cizelgesi oluşturulmuş ve kod blokları ile detaylı olarak açıklanmıştır.[16]

1. **D**eğişkenlerin ve İthalatların Tanımlanması:

    - İlk olarak, kodun başında gerekli kütüphaneler random, numpy, math, ve prettytable gibi, import edilir.
    - Ardından, dönemlerdeki dersleri içeren listeler tanımlanır. Her bir ders listesi, ilgili dönemdeki derslerin adlarını içerir.
    - Popülasyon büyüklüğünü temsil eden 'populasyon', ders sayısını temsil eden 'ders sayisi' ve mutasyona uğrayan birey sayısını belirten 'mutasyona uğrayanlar' gibi değişkenler tanımlanır.
    - Mutasyon olasılığı 'mutasyon olasılığı' ve maksimum nesil sayısı 'maksimum nesil 'değişkenleri belirlenir.

```python

import random
import numpy as np
import math
import prettytable as prettytable
from prettytable import DEFAULT
# Her bir dönemdeki dersleri içeren listeler
dersler_5inci = ["Simülasyon", "Fizik IV", "Fizik V",
                 "Bilgisayar Bilimi", "Kompozisyon",
                  "Elektronik IV", "Boş"]
dersler_6ncı = ["Proje VI", "Elektronik V", "Mekanik VI",
                "Mekanik VII", "Sistemler", "Ölçme", "Boş"]
dersler_7nci = ["Yenilenebilir Enerji", "Bilgisayar Bilimi IV",
                "Mekatronik II", "Proje II", "Elektronik II",
                 "Mikroekonomi", "Boş"]
# Her dersin karşılık geldiği öğretim görevlileri ve haftanın günleri
ogretim_gorevlileri_5inci = {"Elektronik IV": "Grefa", "Kompozisyon": "R. Guachi",
                    "Simülasyon": "R. Guachi", "Fizik IV": "Oscullo",
                     "Bilgisayar Bilimi": "L. Guachi", "Fizik V": "Tirira",
                       "Boş": "Boş"}
ogretim_gorevlileri_6ncı = {"Proje VI": "Duchi", "Elektronik V": "Grefa",
                    "Mekanik VI": "Jacome", "Mekanik VII": "Jacome",
                     "Sistemler": "Velarde", "Ölçme": "Corrales",
                       "Boş": "Boş"}
ogretim_gorevlileri_7nci = {"Yenilenebilir Enerji": "Oscullo",
                    "Bilgisayar Bilimi IV": "Corrales",
                     "Mekatronik II": "Castro", "Proje II": "Duchi",
                       "Elektronik II": "Velarde", "Mikroekonomi": "Altamirano", "Boş": "Boş"}
haftanin_gunleri = {0: "Pazartesi", 1: "Salı", 2: "Çarşamba", 3: "Perşembe", 4: "Cuma"}

```

```python

  # Popülasyon boyutunu belirtin
  populasyon = 8
  # Derslerin sayısı
  ders_sayisi = len(dersler_5inci)
  # Popülasyon boyutunu belirlemek için boş bir liste oluşturun
  Populasyonlar = [0] * populasyon
  Fitness_degeri = [0] * populasyon
  # Mutasyona tabi tutulacak birey sayısı
  mutasyona_uğrayanlar = 4
  # Mutasyon olasılığı
  mutasyon_olasılığı = 0.3
  # Maksimum nesil sayısı
  maksimum_nesil = 100
```

2. **Z**aman Tablosunu Yazdırmak İçin Fonksiyon:

   - **'ders programını yazdır'** fonksiyonu, bir ders programını daha okunaklı bir şekilde yazdırmak için oluşturulmuştur. Bu fonksiyon, **prettytable**" kütüphanesini kullanarak bir tablo oluşturur ve ders programını bu tabloya doldurur.

```
45
46  # Zaman tablosunu daha iyi anlaşılabilir ve düzenli bir şekilde yazdırmak için bir fonksiyon
47  def ders_programını_yazdır(program, birey, ogretim_gorevlileri):
48      tablo = prettytable.PrettyTable()
49      tablo.field_names = ["Pazartesi", "Salı", "Çarşamba", "Perşembe", "Cuma"]
50      for gun in range(7):
51          dersler = [" ", " ", " ", " ", " "]
52          ogretim_gorevlileri_listesi = [" ", " ", " ", " ", " "]
53          for x in range(5):
54              dersler[x] = program[birey][x][gun]
55              ogretim_gorevlileri_listesi[x] = ogretim_gorevlileri.get(dersler[x])
56          tablo.add_row(dersler)
57          tablo.add_row(ogretim_gorevlileri_listesi)
58          tablo.add_row([" ", " ", " ", " ", " "])
59
60      tablo.header = True
61      tablo.horizontal_char = '-'
62      tablo.junction_char = ' '
63      tablo.add_column("Ders Saati", ["7:00 - 9:00", " ", " ", "9:00 - 11:00", " ", " ", "11:00 - 12:00", " ", " ", "12:00 - 13:00",
64                                      " ", " ", "13:00 - 14:00", " ", " ", "14:00 - 16:00", " ", " ", "16:00 - 18:00", " ", " "])
65      print(tablo)
```

3. **G**enetik Algoritmayı Kullanarak En İyi Ders Programını Bulmak İçin Fonksiyonlar:

   - **s**ıralama Fonksiyonu, bir listeyi küçükten büyüğe sıralamak için kullanılır.

```
# En iyi ders programını bulmak için genetik algoritmayı kullan
def siralama_yap(birListe, birListe2):
    for numPasada in range(len(birListe) - 1, 0, -1):
        for i in range(numPasada):
            if birListe[i] > birListe[i + 1]:
                temp = birListe[i]
                temp2 = birListe2[i]
                birListe[i] = birListe[i + 1]
                birListe2[i] = birListe2[i + 1]
                birListe[i + 1] = temp
                birListe2[i + 1] = temp2

    return (birListe, birListe2)
```

```
80
81  # Başlangıç popülasyonunu oluştur
82  def PopulasyonuOlustur(Program_init, dersler):
83      for birey in range(populasyon):
84          for gun in range(5):
85              Program_init[birey][gun] = random.sample(dersler, len(dersler))
86      return Program_init
87
88  # Fitness fonksiyonu
89  def uygunluk_hesapla(Program_5, Program_6, Program_7, birey):
90      çakışmalar = 0
91      for günler in range(5):
92          for ders in range(ders_sayisi):
93              Ogretmen_5 = ogretim_gorevlileri_5inci.get(Program_5[birey][günler][ders])
94              Ogretmen_6 = ogretim_gorevlileri_6ncı.get(Program_6[birey][günler][ders])
95              Ogretmen_7 = ogretim_gorevlileri_7nci.get(Program_7[birey][günler][ders])
96              if (Ogretmen_5 == Ogretmen_6 or Ogretmen_5 == Ogretmen_7 and Ogretmen_5 != "Boş")
97                  çakışmalar += 1
98              if (Ogretmen_6 == Ogretmen_7 and Ogretmen_6 != "Boş"):
99                  çakışmalar += 1
100     return (çakışmalar)
101
```

- **PopulasyonuOlustur** fonksiyonu, başlangıç popülasyonunu oluşturur. Her bir birey, her bir günde rastgele seçilen derslerden oluşur.
- **uygunluk hesapla** fonksiyonu, bir bireyin uygunluğunu (fitness) hesaplar. Bu uygunluk, derslerin çakışma sayısına bağlıdır; yani aynı gün ve saatte aynı öğretmenin iki dersi olması durumunda uygunluk azalır.

```python
# Mutasyon işlemi
def MutasyonaUgrat(program, dersler):
    for gün in range(5):
        if (random.uniform(0, 1) <= mutasyon_olasılığı):
            program[gün] = random.sample(dersler, len(dersler))
    return program

# Yeni nesil oluşturma işlemi
def YeniNesilOlustur(program, populasyonlar, dersler_mut):
    yeni_populasyon = program
    yeni_populasyon[0] = program[populasyonlar[0]]
    yeni_populasyon[1] = program[populasyonlar[1]]
    for bireyler in range(2, len(populasyonlar) - 1, 2):
        kesme_noktası = math.floor(random.uniform(0, 5.9))
        ebeveyn1 = program[populasyonlar[bireyler]]
        ebeveyn2 = program[populasyonlar[bireyler + 1]]
        çocuk1 = ebeveyn1
        çocuk2 = ebeveyn2
        if kesme_noktası == 5:
            çocuk1 = ebeveyn1
            çocuk2 = ebeveyn2
        else:
            gün_ = 0
            while gün_ <= kesme_noktası:
                çocuk1[gün_] = ebeveyn1[gün_]
                çocuk2[gün_] = ebeveyn2[gün_]
                gün_ += 1
            while gün_ < 5:
                çocuk2[gün_] = ebeveyn1[gün_]
                çocuk1[gün_] = ebeveyn2[gün_]
                gün_ += 1
        yeni_populasyon[bireyler] = çocuk1
        yeni_populasyon[bireyler + 1] = çocuk2
        for mutasyon in range(1, mutasyona_uğrayanlar):
            yeni_populasyon[-mutasyon] = MutasyonaUgrat(yeni_populasyon[-mutasyon], dersler_mut)
    return (yeni_populasyon)

```

- **MutasyonaUgrat** fonksiyonu, bir bireye mutasyon uygular. Mutasyon, belirli bir olasılıkla rastgele derslerin yerlerini değiştirerek gerçekleşir.
- **YeniNesilOlustur** fonksiyonu, bir sonraki nesli oluşturur. Bu fonksiyon, melezleme ve mutasyon operatörlerini kullanarak yeni bireyler oluşturur.

```
# Haftalık ders programını oluşturma işlemi
Ders_Programı_5inci = [["0" for i in range(5)] for j in range(populasyon)]
Ders_Programı_6ncı = Ders_Programı_5inci
Ders_Programı_7nci = Ders_Programı_5inci

Ders_Programı_5inci = PopulasyonuOlustur(Ders_Programı_5inci, dersler_5inci)
Ders_Programı_5 = np.array(Ders_Programı_5inci)
Ders_Programı_6ncı = PopulasyonuOlustur(Ders_Programı_6ncı, dersler_6ncı)
Ders_Programı_6 = np.array(Ders_Programı_6ncı)
Ders_Programı_7nci = PopulasyonuOlustur(Ders_Programı_7nci, dersler_7nci)
Ders_Programı_7 = np.array(Ders_Programı_7nci)

nesil_sayacı = 0
while nesil_sayacı < maksimum_nesil:
    if nesil_sayacı % 10 == 0:
        print("Nesil: ", nesil_sayacı)
    for birey in range(populasyon):
        Fitness_degeri[birey] = uygunluk_hesapla(Ders_Programı_5, Ders_Programı_6, Ders_Programı_7, birey - 1)
        Populasyonlar[birey] = birey

        if (Fitness_degeri[birey] == 0):
            print("Çözüm bulundu, öğrenci: ", birey, "\n")
            print("Nesil: ", nesil_sayacı)
            print("5. Yarıyıl Ders Programı\n")
            ders_programını_yazdır(Ders_Programı_5, birey, ogretim_gorevlileri_5inci)
            print("6. Yarıyıl Ders Programı\n")
            ders_programını_yazdır(Ders_Programı_6, birey, ogretim_gorevlileri_6ncı)
            print("7. Yarıyıl Ders Programı\n")
            ders_programını_yazdır(Ders_Programı_7, birey, ogretim_gorevlileri_7nci)

            nesil_sayacı = maksimum_nesil
            break
    Fitness_degeri, Populasyonlar = siralama_yap(Fitness_degeri, Populasyonlar)
    Ders_Programı_5 = YeniNesilOlustur(Ders_Programı_5, Populasyonlar, dersler_5inci)
    Ders_Programı_6 = YeniNesilOlustur(Ders_Programı_6, Populasyonlar, dersler_6ncı)
    Ders_Programı_7 = YeniNesilOlustur(Ders_Programı_7, Populasyonlar, dersler_7nci)
    nesil_sayacı += 1

```

4. **G**enetik Algoritmayı Uygulama ve En İyi Ders Programını Bulma:

    - Her bir matris, bir popülasyon büyüklüğü kadar satıra ve her bir gün için bir sütuna sahiptir. Bu matrisler, başlangıçta rastgele ders programları ile doldurulur.
    - 'while ' döngüsü, maksimum nesil sayısına ulaşılana kadar çalışır. Her nesilde, her bir bireyin uygunluğu hesaplanır ve en uygun bireyler seçilir.
    - Seçilen bireyler arasında çaprazlama ve mutasyon operatörleri uygulanarak yeni bir nesil oluşturulur.
    - Her bir dönem için ayrı ayrı uygunluk hesaplamaları yapılır ve en uygun ders programı bulunduğunda, program durdurulur ve bulunan ders programı ekrana yazdırılır.

```
Nesil:  0
Nesil:  10
Nesil:  20
Nesil:  30
Çözüm bulundu, öğrenci:  4

Nesil:  36
5. Yarıyıl Ders Programı
```

Pazartesi	Salı	Çarşamba	Perşembe	Cuma	Ders Saati
Boş Boş	Fizik V Tirira	Kompozisyon R. Guachi	Fizik V Tirira	Elektronik IV Grefa	7:00 - 9:00
Kompozisyon R. Guachi	Boş Boş	Fizik V Tirira	Fizik IV Oscullo	Boş Boş	9:00 - 11:00
Bilgisayar Bilimi L. Guachi	Fizik IV Oscullo	Bilgisayar Bilimi L. Guachi	Simülasyon R. Guachi	Fizik V Tirira	11:00 - 12:00
Elektronik IV Grefa	Bilgisayar Bilimi L. Guachi	Fizik IV Oscullo	Boş Boş	Simülasyon R. Guachi	12:00 - 13:00
Simülasyon R. Guachi	Simülasyon R. Guachi	Boş Boş	Kompozisyon R. Guachi	Kompozisyon R. Guachi	13:00 - 14:00
Fizik IV Oscullo	Kompozisyon R. Guachi	Simülasyon R. Guachi	Elektronik IV Grefa	Fizik IV Oscullo	14:00 - 16:00
Fizik V Tirira	Elektronik IV Grefa	Elektronik IV Grefa	Bilgisayar Bilimi L. Guachi	Bilgisayar Bilimi L. Guachi	16:00 - 18:00

6. Yarıyıl Ders Programı

Pazartesi	Salı	Çarşamba	Perşembe	Cuma	Ders Saati
Mekanik VII Jacome	Boş Boş	Elektronik V Grefa	Sistemler Velarde	Elektronik V Grefa	7:00 - 9:00
Boş Boş	Ölçme Corrales	Ölçme Corrales	Ölçme Corrales	Mekanik VI Jacome	9:00 - 11:00
Mekanik VI Jacome	Mekanik VI Jacome	Mekanik VI Jacome	Elektronik V Grefa	Proje VI Duchi	11:00 - 12:00
Elektronik V Grefa	Mekanik VII Jacome	Sistemler Velarde	Boş Boş	Mekanik VII Jacome	12:00 - 13:00
Proje VI Duchi	Elektronik V Grefa	Boş Boş	Mekanik VII Jacome	Sistemler Velarde	13:00 - 14:00
Sistemler Velarde	Proje VI Duchi	Mekanik VII Jacome	Proje VI Duchi	Ölçme Corrales	14:00 - 16:00
Ölçme Corrales	Sistemler Velarde	Proje VI Duchi	Mekanik VI Jacome	Boş Boş	16:00 - 18:00

7. Yarıyıl Ders Programı

Pazartesi	Salı	Çarşamba	Perşembe	Cuma	Ders Saati
Mekatronik II Castro	Elektronik II Velarde	Mikroekonomi Altamirano	Elektronik II Velarde	Mikroekonomi Altamirano	7:00 - 9:00
Mikroekonomi Altamirano	Mekatronik II Castro	Yenilenebilir Enerji Oscullo	Bilgisayar Bilimi IV Corrales	Yenilenebilir Enerji Oscullo	9:00 - 11:00
Bilgisayar Bilimi IV Corrales	Proje II Duchi	Bilgisayar Bilimi IV Corrales	Yenilenebilir Enerji Oscullo	Proje II Duchi	11:00 - 12:00
Elektronik II Velarde	Boş Boş	Boş Boş	Boş Boş	Bilgisayar Bilimi IV Corrales	12:00 - 13:00
Yenilenebilir Enerji Oscullo	Bilgisayar Bilimi IV Corrales	Elektronik II Velarde	Mikroekonomi Altamirano	Elektronik II Velarde	13:00 - 14:00
Boş Boş	Yenilenebilir Enerji Oscullo	Proje II Duchi	Mekatronik II Castro	Boş Boş	14:00 - 16:00
Proje II Duchi	Mikroekonomi Altamirano	Mekatronik II Castro	Proje II Duchi	Mekatronik II Castro	16:00 - 18:00

4 Veri Ve Sistem Tasarımı

Bu bölümünde yapılan çalışmada kullanılan veriler ve çalışma mödelinin tasarımı detaylı olarak anlılmıştır.
Çalışmada kullanılan veriler; Ahsen KÜÇÜK'ün hazırlamış olduğu ve Prof. Dr. İpek DEVECİ KOCAKOÇ danişmanlık yaptığı "HEMŞİRE ÇİZELGELEME PROBLEMLERİNİN GENETİK ALGORİTMALARLA OPTİMİZASYONU VE BİR UYGULAMA" başlıklı yüksek lisans tez çalışmasında kullanılan Buca Seyfi Demirsoy Devlet Hastanesi, Koroner Yoğun Bakım servisinin Şubat 2015 çizelgeleridir.[2]

4.1 Hemşire Nöbet Çizelgeleme Problemi

Hemşire nöbet çizelgesi problemi, sağlık sektöründe kaynakların verimli kullanımı ve kaliteli hasta bakımı sağlanması açısından büyük önem taşıyan bir çizelgeleme problemidir. Bu problem, hemşirelerin vardiya saatlerinin düzenlenmesi ve hastanenin 24 saat kesintisiz hizmet vermesini sağlamak amacıyla, karmaşık kısıtlar ve hedeflerin optimize edilmesini gerektirir.

Hemşire nöbet çizelgesi problemi, sağlık hizmetlerinin sürekliliğini ve kalitesini sağlamak için kritik bir öneme sahiptir. Bu problemin etkin bir şekilde çözülmesi, hemşirelerin iş yükünü dengelerken hastaların ihtiyaç duyduğu kesintisiz bakımı sağlamaya yardımcı olur. Optimizasyon teknikleri ve özellikle genetik algoritmalar, bu süreçte önemli bir rol oynayarak, hastane yönetimlerine ve sağlık çalışanlarına büyük faydalar sağlar.

4.2 Hemşire Nöbet Cizelgeleme Problemi Test Mödeli

Bölümde Genetik algoritmalar ile bir Test modeli tasarlanmıştır.Test modelimizde 5 hemşirenin, 7 günlük içinde ve 2 vardiya şeklinde nöbet çizelgesi oluşturulmuş ve tablo şeklinde gösterilmiştir.Amaç modelimizin çalışmasını ve sonuclarını gözlemlemektir.Bu sayede sistemin Gerçek mödel de daha iyi çalışması amaçlanmıştır.

4.2.1 Parametrelerin Tanımlanması:

n, m, k, u, ve h gibi parametreler belirtilir. Bunlar sırasıyla hemşire sayısı, gün sayısı, vardiya sayısı, uzman hemşire sayısı ve gece nöbetine

kalmayacak hemşire sayısıdır.

```python
import numpy as np
from scipy.optimize import minimize

class GenetikAlgoritma:
    def __init__(self, n, m, k, u, h):
        self.n = n  # Hemşire sayısı
        self.m = m  # Gün sayısı
        self.k = k  # Vardiya sayısı
        self.u = u  # Uzman hemşire sayısı
        self.h = h  # Gece nöbetine kalmayacak hemşire sayısı
        self.populasyon_boyutu = 2  # Popülasyon boyutu
        self.maksimum_nesil = 10  # Maksimum nesil sayısı
        self.mutasyon_orani = 1  # Mutasyon oranı
        self.populasyon = self.populasyon_olustur()

    def amac_fonksiyonu(self, X):
        """
        Amaç fonksiyonu: Çalışma sürelerinin minimizasyonu
        """
        fmin = 0
        for i in range(self.n):
            for d in range(self.m):
                fmin += 8 * X[i*self.m + d] + 16* X[self.n*self.m + i*self.m + d]
        return fmin

```

4.2.2 Genetik Algoritmanın Oluşturulması

GenetikAlgoritma adında bir sınıf tanımlanır. Bu sınıf, genetik algoritmanın bütün özelliklerini içerir.

4.2.3 Amaç Fonksiyonunun Tanımlanması:

'amac fonksiyonu' metodu, bireylerin uygunluk değerlerini hesaplar. Bu uygunluk değerleri, çalışma sürelerinin minimize edilmesine yönelik bir amaç fonksiyonu kullanılarak elde edilir.Amaç fonksiyonu, her bir bireyin ne kadar iyi olduğunu ölçer, her bir bireyin toplam çalışma süresi hesaplanır.

4.2.4 Başlangıç Popülasyonunun Oluşturulması:

'populasyon olustur' metodu kullanılarak başlangıç popülasyonu oluşturulur. Her birey, bir vardiyadaki her hemşirenin her gün için bir

vardiya yapmasını temsil eden bir genotip içerir. Genetik algoritma bir başlangıç popülasyonu ile başlar. Her bir birey, problemdeki bir çözümü temsil eder.Her bir birey bir hemşirenin her bir gün için vardiya yapma durumunu temsil eder. Başlangıç popülasyonu, rastgele seçilen genlerden oluşur.

```
def populasyon_olustur(self):
    """
    Başlangıç populasyonunu oluştur
    """
    return np.random.randint(2, size=(self.populasyon_boyutu, self.n*(self.m*2)))

def uygunluk_hesapla(self):
    """
    Popülasyon için uygunluk değerlerini hesapla
    """
    uygunluk_degerleri = []
    for birey in self.populasyon:
        uygunluk_degerleri.append(self.amac_fonksiyonu(birey))
    return uygunluk_degerleri

def sec(self, uygunluk_degerleri):
    """
    Seçim işlemi
    """
    return np.random.choice(range(len(self.populasyon)), size=self.populasyon_boyutu,
                            replace=True, p=(uygunluk_degerleri / np.sum(uygunluk_degerleri)))
```

4.2.5 Seçim İşlemi:

Popülasyon içindeki bireylerin uygunluk değerlerine göre seçim yapılır. Daha uygun bireyler, daha büyük bir olasılıkla seçilir. Bu, daha iyi çözümlerin gelecek nesillere aktarılmasını sağlar.

```
def sec(self, uygunluk_degerleri):
    """
    Seçim işlemi
    """
    secim_olasiliklari = np.array(uygunluk_degerleri) / np.sum(uygunluk_degerleri)
    return np.random.choice(range(len(self.populasyon)), size=self.populasyon_boyutu,
                            replace=True, p=secim_olasiliklari)
```

4.2.6 Kısıtların Tanımlanması

kisitlar metodu, problemdeki kısıtları tanımlar.. Her hemşirenin bir günde en fazla bir vardiya yapması, uzman hemşirelerin her gün en az bir vardiya yapması ve gece nöbetine kalmayacak hemşirelerin her gün gece vardiyası yapmaması gibi kısıtlar belirlenir.

```
def kisitlar(self, X):
    """
    Kısıtlar
    """
    kisitlar = []
    # Kisit 1: Her hemşirenin bir günde en fazla bir vardiya yapması
    for i in range(self.n):
        for d in range(self.m):
            kisitlar.append(X[i*self.m + d] + X[self.n*self.m + i*self.m + d] <= 1)
    # Kisit 2: Uzman hemşirelerin her gün en az bir vardiya yapması
    for d in range(self.m):
        uzman_toplam = 0
        for i in range(self.u):
            uzman_toplam += X[self.n*self.m + i*self.m + d]
        kisitlar.append(uzman_toplam >= 1)
    # Kisit 3: Gece nöbetine kalmayacak hemşirelerin her gün gece vardiyası yapmaması
    for d in range(self.m):
        gece_toplam = 0
        for i in range(self.h):
            gece_toplam += X[self.n*self.m + self.u*self.m + i*self.m + d]
        kisitlar.append(gece_toplam == 0)
    return kisitlar
```

Problemdeki kısıtlar, hemşirelerin çalışma düzenini belirler

4.2.7 Çaprazlama İşlemi ve Mutasyon İşlemi:

'caprazla' metodu, seçilen bireyler arasında çaprazlama işlemi gerçekleştirilir. Bu işlem, iki bireyin genetik materyalinin bir kısmının birbiriyle değiştirilmesini sağlar. Yeni bireyler, ebeveynlerin özelliklerini taşıyan ancak farklılıklar içeren çocuklar olur.
'mutasyon' metodu, Yeni bireylerde mutasyon işlemi uygulanır. Bu, rastgele seçilen genlerin değerlerinin değiştirilmesini içerir. Bu, çeşitliliği arttırır ve potansiyel olarak daha iyi çözümlerin bulunmasına yardımcı olur..

```
def caprazla(self, ebeveyn1, ebeveyn2):
    """
    Çaprazlama işlemi
    """
    nokta = np.random.randint(len(ebeveyn1))
    cocuk1 = np.concatenate((ebeveyn1[:nokta], ebeveyn2[nokta:]))
    cocuk2 = np.concatenate((ebeveyn2[:nokta], ebeveyn1[nokta:]))
    return cocuk1, cocuk2

def mutasyon(self, birey):
    """
    Mutasyon işlemi
    """
    for i in range(len(birey)):
        if np.random.rand() < self.mutasyon_orani:
            birey[i] = 1 - birey[i]
    return birey
```

Yeni nesil bireylerin oluşturulması için seçim, çaprazlama ve mutasyon işlemleri uygulanır

4.2.8 Optimizasyonun Gerçekleştirilmesi:

'optimize et' metodu, genetik algoritmayı belirli bir maksimum nesil sayısına kadar çalıştırır ve en iyi çözümü bulur.

```
def optimize_et(self):
    """
    Genetik algoritmayı çalıştır ve en iyi çözümü bul
    """
    for nesil in range(self.maksimum_nesil):
        uygunluk_degerleri = self.uygunluk_hesapla()
        yeni_populasyon = []

        for _ in range(self.populasyon_boyutu // 2):
            secilenler = self.sec(uygunluk_degerleri)
            ebeveyn1, ebeveyn2 = self.populasyon[secilenler[0]], self.populasyon[secilenler[1]]
            cocuk1, cocuk2 = self.caprazla(ebeveyn1, ebeveyn2)
            cocuk1 = self.mutasyon(cocuk1)
            cocuk2 = self.mutasyon(cocuk2)
            yeni_populasyon.append(cocuk1)
            yeni_populasyon.append(cocuk2)

        self.populasyon = np.array(yeni_populasyon)

        if nesil % 10 == 0:
            print("Nesil {}: En iyi çözüm: {}".format(nesil, min(uygunluk_degerleri)))

    en_iyi_birey = self.populasyon[np.argmin(uygunluk_degerleri)]
    en_iyi_cozum = self.amac_fonksiyonu(en_iyi_birey)
    print("En iyi çözüm:", en_iyi_cozum)
    return en_iyi_cozum, en_iyi_birey
```

Belirli bir nesil sayısı veya başka bir durum sağlandığında optimizasyon işlemi sonlandırılır. En iyi çözüm, amaç fonksiyonunun en küçük olduğu birey olarak belirlenir.

4.2.9 Sonuçların Gösterilmesi:

En iyi çözüm ve uygunluk değerleri gösterilir. Bu, problemdeki en iyi çözümün ne olduğunu ve bu çözümün ne kadar iyi olduğunu belirtir.

```
# Genetik Algoritma'nın kullanılması
genetik_algoritma = GenetikAlgoritma(n, m, k, u, h)
genetik_algoritma.optimize_et()
```

4.2.10 Cıktı

Belirtilen kısıtlamalar ve amac fonksiyonu ile hemşirelerin calışma saatlerini minimuma indirip ,oluşturulabilecek en optimum sonucu burada gösterilmektedir.

```
Nesil 0: En iyi çözüm: 256
Nesil 10: En iyi çözüm: 360
Nesil 20: En iyi çözüm: 368
Nesil 30: En iyi çözüm: 376
Nesil 40: En iyi çözüm: 384
Nesil 50: En iyi çözüm: 384
Nesil 60: En iyi çözüm: 368
Nesil 70: En iyi çözüm: 368
Nesil 80: En iyi çözüm: 392
Nesil 90: En iyi çözüm: 392
En iyi çözüm: 400
Hemşire Çizelgesi (Gündüz: 1, Gece: 2):
```

Table 1: Test Çalışması Sonuç Tablosu

Hemşire	Tarih	Vardiya
Hemşire 1	2024-05-30 Perşembe	İzinli
Hemşire 2	2024-05-30 Perşembe	Gece
Hemşire 3	2024-05-30 Perşembe	Gündüz
Hemşire 4	2024-05-30 Perşembe	İzinli
Hemşire 5	2024-05-30 Perşembe	Gece
Hemşire	Tarih	Vardiya
Hemşire 1	2024-05-31 Cuma	Gündüz
Hemşire 2	2024-05-31 Cuma	İzinli
Hemşire 3	2024-05-31 Cuma	Gündüz
Hemşire 4	2024-05-31 Cuma	Gece
Hemşire 5	2024-05-31 Cuma	İzinli
Hemşire	Tarih	Vardiya
Hemşire 1	2024-06-01 Cumartesi	Gündüz
Hemşire 2	2024-06-01 Cumartesi	Gece
Hemşire 3	2024-06-01 Cumartesi	Gündüz
Hemşire 4	2024-06-01 Cumartesi	İzinli
Hemşire 5	2024-06-01 Cumartesi	Gece
Hemşire	Tarih	Vardiya
Hemşire 1	2024-06-02 Pazar	Gündüz
Hemşire 2	2024-06-02 Pazar	Gece
Hemşire 3	2024-06-02 Pazar	Gündüz
Hemşire 4	2024-06-02 Pazar	Gece
Hemşire 5	2024-06-02 Pazar	İzinli
Hemşire	Tarih	Vardiya
Hemşire 1	2024-06-03 Pazartesi	Gece
Hemşire 2	2024-06-03 Pazartesi	İzinli
Hemşire 3	2024-06-03 Pazartesi	Gündüz
Hemşire 4	2024-06-03 Pazartesi	İzinli
Hemşire 5	2024-06-03 Pazartesi	Gece
Hemşire	Tarih	Vardiya
Hemşire 1	2024-06-04 Salı	İzinli
Hemşire 2	2024-06-04 Salı	Gece
Hemşire 3	2024-06-04 Salı	İzinli
Hemşire 4	2024-06-04 Salı	Gündüz
Hemşire 5	2024-06-04 Salı	Gece
Hemşire	Tarih	Vardiya
Hemşire 1	2024-06-05 Çarşamba	Gece
Hemşire 2	2024-06-05 Çarşamba	İzinli
Hemşire 3	2024-06-05 Çarşamba	Gece
Hemşire 4	2024-06-05 Çarşamba	Gündüz
Hemşire 5	2024-06-05 Çarşamba	İzinli

4.3 Hemşire Nöbet Çizelgeleme Problemi Model Sistem Tasarımı

4.3.1 Parametrelerin Tanımlanması:

Hemşire nöbet çizelgesi probleminin başarılı bir şekilde çözülmesi için, ilgili parametrelerin dikkatlice tanımlanması gerekmektedir. Parametreler, problemin yapısını ve çözüm sürecini belirleyen temel unsurlardır.

Çalışmada n, m, k, u, ve h gibi parametreler belirtilir. Bunlar sırasıyla hemşire sayısı, gün sayısı, vardiya sayısı, uzman hemşire sayısı ve gece nöbetine kalmayacak hemşire sayısıdır.

GenetikAlgoritma sınıfı, hemşire vardiyası optimizasyon problemi için bir çerçeve sağlar. Bu sınıf, problemle ilgili parametreleri ve yöntemleri içerir.

'init' yöntemi, sınıfın başlatıcı yöntemidir. Bu yöntem, problemde kullanılacak parametreleri alır ve başlangıç popülasyonunu oluşturur.

```python
import numpy as np

class GenetikAlgoritma:
    def __init__(self, n, m, k, u, h):
        self.n = n  # Hemşire sayısı
        self.m = m  # Gün sayısı
        self.k = k  # Vardiya sayısı
        self.u = u  # Uzman hemşire sayısı
        self.h = h  # Gece nöbetine kalmayacak hemşire sayısı
        self.populasyon_boyutu = 50  # Popülasyon boyutu
        self.maksimum_nesil = 100  # Maksimum nesil sayısı
        self.mutasyon_orani = 0.01  # Mutasyon oranı
        self.populasyon = self.populasyon_olustur()

    def populasyon_olustur(self):
        """
        Başlangıç popülasyonunu oluştur
        """
        return np.random.randint(2, size=(self.populasyon_boyutu, self.n * self.m * 2))
```

4.3.2 Amaç Fonksiyonunun Tanımlanması:

Amaç fonksiyonu, çizelgeleme problemlerinin çözümünde belirli bir hedefe ulaşmayı sağlayan matematiksel bir ifadedir. Bu fonksiyon, problemin çözüm sürecinde hangi kriterlerin optimize edileceğini belirler.

Bu fonksiyon, çözümün etkinliğini ve başarısını belirleyen temel unsurlardan biridir. Doğru ve uygun bir amaç fonksiyonunun tanımlanması, problemin istenilen şekilde çözülmesini sağlar ve elde edilen çözümlerin kalite seviyesini artırır. Dolayısıyla, çizelgeleme algoritmalarının tasarımında amaç fonksiyonunun dikkatlice seçilmesi ve optimize edilmesi gerekmektedir.

Çalışmamız da 'amac fonksiyonu' metodu, bireylerin uygunluk değerlerini hesaplar. Bu uygunluk değerleri, çalışma sürelerinin minimize edilmesine yönelik bir amaç fonksiyonu kullanılarak elde edilir.Amaç fonksiyonu, her bir bireyin ne kadar iyi olduğunu ölçer, her bir bireyin toplam çalışma süresi hesaplanır.

$$\text{Minimize} \quad \sum_{i=0}^{n-1} \sum_{d=0}^{m-1} (8 \cdot X_{i \cdot m+d} + 16 \cdot X_{n \cdot m+i \cdot m+d}) \tag{1}$$

Denklem 1 ,Çalışma sürelerinin en aza indirilmesi sağlamaktadır.

```
def amac_fonksiyonu(self, X):
    """
    Amaç fonksiyonu: Çalışma sürelerinin minimizasyonu
    """
    fmin = 0
    for i in range(self.n):
        for d in range(self.m):
            fmin += 8 * X[i*self.m + d] + 16 * X[self.n*self.m + i*self.m + d]
    return fmin
```

4.3.3 Kısıtların Tanımlanması:

Kısıtlar, belirli görevlerin veya işlemlerin yerine getirilmesi sırasında karşılaşılan sınırlamaları ve zorunlulukları ifade eder.

Kısıtların önemi, çizelgeleme problemlerinin çözüm sürecinde ortaya çıkar. Kısıtlar, çözümün geçerliliğini belirleyen temel unsurlar olduğundan, bu kısıtların ihlal edilmesi durumunda elde edilen çözümler pratikte uygulanamaz hale gelir. Dolayısıyla, çizelgeleme algoritmalarının tasarımında kısıtların dikkatlice ele alınması ve çözüm sürecine entegre edilmesi gerekmektedir.

Çalışmamız da "kisitlar" fonksiyonu ile problemdeki kısıtları tanımlar. Problemdeki kısıtlar, hemşirelerin çalışma düzenini belirler.
Bu kısıtlar, her hemşirenin günlük maksimum vardiya sayısı, uzman hemşirelerin her gün en az bir vardiya yapması gibi kuralları içerir.

Uygulamada kullanılan kısıtların denklem karşılıkları aşağıda verilmiştir

$$X_{i,d} + X_{n \cdot m+i \cdot m+d} \leq 1 \quad \forall i, \forall d \tag{2}$$

Denklem 2 ,Her hemşire bir günde en fazla bir vardiya yapabilir kısıtını sağlamaktadır

$$\sum_{i=0}^{u-1} X_{n \cdot m+i \cdot m+d} \geq 1 \quad \forall d \tag{3}$$

Denklem 3 ,Uzman hemşirelerin her gün en az bir vardiya yapması kısıtını sağlamaktadır

$$\sum_{i=0}^{h-1} X_{n \cdot m+u \cdot m+i \cdot m+d} = 0 \quad \forall d \tag{4}$$

Denklem 4, Gece nöbetine kalmayacak hemşirelerin her gün gece vardiyası yapmaması kısıtını sağlamaktadır

$$X_{i,d} + X_{n \cdot m+i \cdot m+d} \leq 1 \quad \forall i, \forall d \tag{5}$$

Denklem 5, Gündüz vardiyasında çalışan bir hemşirenin gece vardiyasında çalışmaması kısıtını sağlamaktadır

$$\sum_{i=0}^{n-1} X_{n \cdot m+i \cdot m+d} = 2 \quad \forall d \tag{6}$$

Denklem 6 ,Her hemşire nöbetçi olmadığı gün 8 saat çalışır. Haftada 40 saat, ayda 160 saat çalışma zorunluluğu kısıtını sağlamaktadır

$$\sum_{d=0}^{m-1} [8(1 - X_{i,d}) + 16X_{n \cdot m+i \cdot m+d}] = saat \quad \forall i \tag{7}$$

Denklem 7, Akşam nöbeti tutanlara ertesi gün görev verilmez kısıtını sağlamaktadır

$$X_{i,d} + X_{n \cdot m+i \cdot m+d} \leq 1 \quad \forall i, \forall d \tag{8}$$

Denklem 8, Hemşire aynı gün içinde yalnızca bir vardiyada çalışabilir ya da izinlidir kısıtını sağlamaktadır

$$X_{i,d} + X_{n \cdot m+i \cdot m+d} \leq 1 \quad \forall i, \forall d \tag{9}$$

Denklem 9, Akşam vardiyasında nöbet tutan iki kişiden birinin uzman olması kısıtını sağlamaktadır

$$\sum_{i=0}^{u-1} X_{n \cdot m+i \cdot m+d} \geq 1 \quad \forall d \tag{10}$$

Denklem 10 , Sabah vardiyasında çalışan hemşire akşam vardiyasında çalışmayacak kısıtını sağlamaktadır

$$X_{i,d} + X_{n \cdot m+i \cdot m+d} \leq 1 \quad \forall i, \forall d$$

Denklem 4.3.3, Uzman hemşireler gece vardiyasında en az bir kişi olmalı kısıtını sağlamaktadır

$$\sum_{i=0}^{u-1} X_{n \cdot m+i \cdot m+d} \geq 1 \quad \forall d \tag{11}$$

Denklem 11, Gece nöbetine kalmayacak hemşireler kısıtını sağlamaktadır

$$\sum_{i=0}^{h-1} X_{n \cdot m+u \cdot m+i \cdot m+d} = 0 \quad \forall d \tag{12}$$

Denklem 12 ,Gece nöbetinde iki kişi çalışır (kısıtlı olanlar hariç) kısıtını sağlamaktadır

$$\sum_{i=0}^{n-1} X_{n \cdot m+i \cdot m+d} = 2 \quad \forall d \tag{13}$$

Denklem 13, Ayda en az 160 saat calışma kısıtını sağlayan denklem kısıtını sağlamaktadır

$$\sum_{d=0}^{m-1} [8X_{i,d} + 16X_{n \cdot m+i \cdot m+d}] \geq 160 \quad \forall i \tag{14}$$

Denklem 14, Ayda en az 160 saat çalışmalı kısıtını sağlayan denklem kısıtını ssağlamaktadır

$$X_{n \cdot m+i \cdot m+d} + X_{i,d+1} = 0 \quad \forall i, \forall d < m \tag{15}$$

Denklem 15 ,Gece nöbet tutan ertesi gün gündüz çalışmaz kısıtını sağlayan denklem kısıtını sağlamaktadır

Extra olarak uygulamada kullanılan kısıtların kod satırlarının resimleri aşağıda verilmiştir.Her bir kısıt için yorum satırları oluşturulmuş ve bilgisi verilmiştir.

```python
def kisitlar(self, X):
    """
    Kısıtlar
    """
    kisitlar = []
    # Kisit 1: Her hemşirenin bir günde en fazla bir vardiya yapması
    for i in range(self.n):
        for d in range(self.m):
            kisitlar.append(X[i*self.m + d] + X[self.n*self.m + i*self.m + d] <= 1)

    # Kisit 2: Uzman hemşirelerin her gün en az bir vardiya yapması
    for d in range(self.m):
        uzman_toplam = 0
        for i in range(self.u):
            uzman_toplam += X[self.n*self.m + i*self.m + d]
        kisitlar.append(uzman_toplam >= 1)

    # Kisit 3: Gece nöbetine kalmayacak hemşirelerin her gün gece vardiyası yapmaması
    for d in range(self.m):
        gece_toplam = 0
        for i in range(self.h):
            gece_toplam += X[self.n*self.m + self.u*self.m + i*self.m + d]
        kisitlar.append(gece_toplam == 0)

    # Kisit 4: Gündüz vardiyasında çalışan bir hemşirenin gece vardiyasında çalışmaması
    for i in range(self.n):
        for d in range(self.m):
            kisitlar.append(X[i*self.m + d] + X[self.n*self.m + i*self.m + d] <= 1)

    # Kisit 5: Akşam nöbetinde iki kişi çalışır
    for d in range(self.m):
        aksam_toplam = 0
        for i in range(self.n):
            aksam_toplam += X[self.n*self.m + i*self.m + d]
        kisitlar.append(aksam_toplam == 2)

    # Kisit 6: Her hemşire nöbetçi olmadığı gün 8 saat çalışır. Haftada 40 saat, ayda 160 saat çalışma zorunluluğu
    saat = 40 if self.n == 5 else 160
    for i in range(self.n):
        toplam_saat = 0
        for d in range(self.m):
            toplam_saat += 8 * (1 - X[i*self.m + d]) + 16 * X[self.n*self.m + i*self.m + d]
        kisitlar.append(toplam_saat == saat)

    # Kisit 7: Akşam nöbeti tutanlara ertesi gün görev verilmez
    for i in range(self.n):
        for d in range(self.m - 1):
            kisitlar.append(X[self.n*self.m + i*self.m + d] + X[i*self.m + d + 1] == 0)

    # Kisit 8: Hemşire aynı gün içinde yalnızca bir vardiyada çalışabilir ya da izinlidir
    for i in range(self.n):
        for d in range(self.m):
            kisitlar.append(X[i*self.m + d] + X[self.n*self.m + i*self.m + d] <= 1)

    # Kisit 9: Akşam vardiyasında nöbet tutan iki kişiden birinin uzman olması
    for d in range(self.m):
        uzman_var = 0
        for i in range(self.u):
            uzman_var += X[self.n*self.m + i*self.m + d]
        kisitlar.append(uzman_var >= 1)
```

```
# Kisit 10: Sabah vardiyasında çalışan hemşire akşam vardiyasında çalışmayacak
for i in range(self.n):
    for d in range(self.m):
        kisitlar.append(X[i*self.m + d] + X[self.n*self.m + i*self.m + d] <= 1)

# Kisit 11:Uzman hemşireler gece vardiyasında en az bir kişi olmalı
for d in range(self.m):
    uzman_toplam = 0
    for i in range(self.u):
        uzman_toplam += X[self.n*self.m + i*self.m + d]
    kisitlar.append(uzman_toplam >= 1)

# Kisit 12:Gece nöbetine kalmayacak hemşireler
for d in range(self.m):
    gece_toplam = 0
    for i in range(self.h):
        gece_toplam += X[self.n*self.m + self.u*self.m + i*self.m + d]
    kisitlar.append(gece_toplam == 0)

#Kisit 13:Gece nöbetinde iki kişi çalışır (kısıtlı olanlar hariç)
for d in range(self.m):
    aksam_toplam = 0
    for i in range(self.n):
        aksam_toplam += X[self.n*self.m + i*self.m + d]
    kisitlar.append(aksam_toplam == 2)
```

```
#Kisit 14: Ayda en az 160 saat çalışmalı
saat = 160
for i in range(self.n):
    toplam_saat = 0
    for d in range(self.m):
        toplam_saat += 8 * X[i*self.m + d] + 16 * X[self.n*self.m + i*self.m + d]
    kisitlar.append(toplam_saat >= saat)

# Kisit 15:Gece nöbet tutan ertesi gün gündüz çalışmaz
for i in range(self.n):
    for d in range(self.m - 1):
        kisitlar.append(X[self.n*self.m + i*self.m + d] + X[i*self.m + d + 1] == 0)

return kisitlar
```

4.3.4 Uygunluk Değerlerinin Hesaplanması:

Genetik algoritmalar, çizelgeleme problemleri gibi karmaşık optimizasyon problemlerinin çözümünde etkili bir yöntem olarak öne çıkmaktadır. Bu algoritmaların başarısında uygunluk değeri (fitness value) önemli bir rol oynar. Uygunluk değeri, her bir çözüm adayının (bireyin) problemde belirlenen amaç fonksiyonuna ne kadar iyi uyduğunu ölçen bir metriktir

Genetik algoritmalarda uygunluk değeri, çözüm sürecinin merkezinde yer alır. Uygunluk değeri, çözümlerin kalitesini ölçer, doğal seleksiyon sürecine rehberlik eder, genetik çeşitliliği korur ve optimizasyon hedeflerini dengelemeye yardımcı olur. Bu nedenle, uygunluk değeri, genetik algoritmaların etkinliğini ve başarısını belirleyen kritik bir faktördür. Çizelgeleme problemlerinin çözümünde, uygunluk değerinin doğru ve etkili bir şekilde tanımlanması, algoritmanın performansını ve elde edilen çözümlerin kalitesini önemli ölçüde artırır.

Çalışmamız da "uygunluk hesapla yöntemi", her bireyin uygunluk değerini hesaplar. Bu değerler, bireylerin amaca ne kadar uygun olduğunu gösterir.

```
def uygunluk_hesapla(self):
    """
    Popülasyon için uygunluk değerlerini hesapla
    """
    uygunluk_degerleri = []
    for birey in self.populasyon:
        uygunluk_degerleri.append(self.amac_fonksiyonu(birey))
    return uygunluk_degerleri
```

4.3.5 Seçim İşlemi

Genetik algoritmalarda seçim işlemi (selection process) büyük bir öneme sahiptir çünkü bu işlem, hangi bireylerin bir sonraki nesilde daha fazla temsil edileceğini belirler. Seçim işlemi, popülasyonun genel kalitesini artırarak optimizasyon sürecinin etkinliğini ve hızını doğrudan etkiler. Ayrıca, optimizasyon hedeflerine ulaşmada ve dengeli çözümler bulmada önemli bir rol oynar. Bu nedenle, genetik algoritmaların başarısı ve etkinliği büyük ölçüde seçim işleminin doğru ve etkili bir şekilde gerçekleştirilmesine bağlıdır.

Çalışmamız da "sec yöntemi", seçim işlemini gerçekleştirir. Bu işlem, uygunluk değerlerine göre bireyleri seçer. Daha uygun bireylerin seçilme olasılığı daha yüksektir.

```
def sec(self, uygunluk_degerleri):
    """
    Seçim işlemi
    """
    secim_olasiliklari = np.array(uygunluk_degerleri) / np.sum(uygunluk_degerleri)
    return np.random.choice(range(len(self.populasyon)), size=self.populasyon_boyutu,
                            replace=True, p=secim_olasiliklari)
```

4.3.6 Çaprazlama İşlemi ve Mutasyon İşlemi:

Çaprazlama ve mutasyon, genetik algoritmaların temel bileşenleri olup, optimizasyon süreçlerinde genetik çeşitliliği artırarak daha iyi çözümler bulunmasını sağlar. Çaprazlama, genetik materyalin karıştırılması yoluyla yeni çözümler üretirken, mutasyon genetik çeşitliliği koruyarak ve lokal minimumlardan kaçınarak çözüm alanının genişlemesine yardımcı olur. Bu işlemler, genetik algoritmaların etkinliğini ve başarısını önemli ölçüde artırır.

Çalışmamız da "caprazla" metodu, seçilen bireyler arasında çaprazlama işlemi gerçekleştirilir. Bu işlem, iki bireyin genetik materyalinin bir kısmının birbiriyle değiştirilmesini sağlar. Yeni bireyler, ebeveynlerin özelliklerini taşıyan ancak farklılıklar içeren çocuklar olur.

Ayrıca "mutasyon" metodu ile Yeni bireylerde mutasyon işlemi uygulanır. Bu, rastgele seçilen genlerin değerlerinin değiştirilmesini içerir. Bu, çeşitliliği arttırır ve potansiyel olarak daha iyi çözümlerin bulunmasına yardımcı olur.

Yeni nesil bireylerin oluşturulması için seçim, çaprazlama ve mutasyon işlemleri uygulanır

```python
def caprazla(self, ebeveyn1, ebeveyn2):
    """
    Çaprazlama işlemi
    """
    nokta = np.random.randint(len(ebeveyn1))
    cocuk1 = np.concatenate((ebeveyn1[:nokta], ebeveyn2[nokta:]))
    cocuk2 = np.concatenate((ebeveyn2[:nokta], ebeveyn1[nokta:]))
    return cocuk1, cocuk2

def mutasyon(self, birey):
    """
    Mutasyon işlemi
    """
    for i in range(len(birey)):
        if np.random.rand() < self.mutasyon_orani:
            birey[i] = 1 - birey[i]
    return birey
```

4.3.7 Optimizasyonun Gerçekleştirilmesi:

Optimizasyon, belirli bir hedefe ulaşmak için en iyi çözümün bulunmasını amaçlayan matematiksel ve hesaplamalı bir süreçtir. Genetik algoritmalar, çizelgeleme problemlerinin optimizasyonunda kritik bir role sahiptir. Karmaşık ve çok boyutlu problemlerin çözümünde, esnek ve adaptif yapılarıyla, global optimuma ulaşma yetenekleriyle ve çok kriterli optimizasyon kapasiteleriyle öne çıkarlar.

Çalışmamız da "optimize et" metodu, genetik algoritmayı belirli bir maksimum nesil sayısına kadar çalıştırır ve en iyi çözümü bulur.

Ayrıca "optimize et" yöntemi, genetik algoritmayı çalıştırır. Bu yöntem, belirli bir nesil sayısı boyunca seçim, çaprazlama ve mutasyon işlemlerini tekrarlar. Her iterasyonda, uygunluk değerleri daha iyiye doğru ilerler.

```python
def optimize_et(self):
    """
    Genetik algoritmayı çalıştır ve en iyi çözümü bul
    """
    for nesil in range(self.maksimum_nesil):
        uygunluk_degerleri = self.uygunluk_hesapla()
        yeni_populasyon = []

        for _ in range(self.populasyon_boyutu // 2):
            secilenler = self.sec(uygunluk_degerleri)
            ebeveyn1, ebeveyn2 = self.populasyon[secilenler[0]], self.populasyon[secilenler[1]]
            cocuk1, cocuk2 = self.caprazla(ebeveyn1, ebeveyn2)
            cocuk1 = self.mutasyon(cocuk1)
            cocuk2 = self.mutasyon(cocuk2)
            yeni_populasyon.append(cocuk1)
            yeni_populasyon.append(cocuk2)

        self.populasyon = np.array(yeni_populasyon)

        if nesil % 10 == 0:
            print("Nesil {}: En iyi çözüm: {}".format(nesil, min(uygunluk_degerleri)))

    en_iyi_birey = self.populasyon[np.argmin(uygunluk_degerleri)]
    en_iyi_cozum = self.amac_fonksiyonu(en_iyi_birey)
    print("En iyi çözüm:", en_iyi_cozum)
    return en_iyi_cozum, en_iyi_birey
```

Belirli bir nesil sayısı veya başka bir durum sağlandığında optimizasyon işlemi sonlandırılır. En iyi çözüm, amaç fonksiyonunun en küçük olduğu birey olarak belirlenir.

4.3.8 Sonuçların Gösterilmesi:

En iyi çözüm ve uygunluk değerleri gösterilir. Bu, problemdeki en iyi çözümün ne olduğunu ve bu çözümün ne kadar iyi olduğunu belirtir.

```
# Genetik Algoritma'nın kullanılması
genetik_algoritma = GenetikAlgoritma(n, m, k, u, h)
genetik_algoritma.optimize_et()
```

4.3.9 Cıktı

Belirtilen kısıtlamalar ve amac fonksiyonu ile hemşirelerin calışma saatlerini minimuma indirip ,oluşturulabilecek en optimum sonucu burada gösterilmektedir.

Uygunluk değeri en yüksek olan birey cıktı olarak alınır .

Sonuçlar bir tablo şeklinde extra olarak gösterilmektedir.

```
Nesil 0: En iyi çözüm: 4376
Nesil 10: En iyi çözüm: 4376
Nesil 20: En iyi çözüm: 4440
Nesil 30: En iyi çözüm: 4352
Nesil 40: En iyi çözüm: 4352
Nesil 50: En iyi çözüm: 4248
Nesil 60: En iyi çözüm: 4344
Nesil 70: En iyi çözüm: 4272
Nesil 80: En iyi çözüm: 4152
Nesil 90: En iyi çözüm: 4360
En iyi çözüm: 4944
01.02.2015 Pazar'dan başlayan 28 günlük hemşire nöbet çizelgesi:

Hafta 1:

Hemşireler 01.02.2015 02.02.2015 03.02.2015 04.02.2015 05.02.2015 06.02.2015 07.02.2015
 Hemşire 1     izinli      08`16     izinli      08`16      08`16      08`16      16`08
 Hemşire 2     izinli      08`16      16`08      08`16      08`16      08`16      08`16
 Hemşire 3      08`16      16`08      08`16      08`16      16`08      16`08      08`16
 Hemşire 4      08`16      08`16     izinli      16`08      08`16     izinli     izinli
 Hemşire 5      16`08      16`08     izinli      08`16      08`16     izinli      08`16
 Hemşire 6     izinli      16`08     izinli      08`16     izinli      08`16     izinli
 Hemşire 7      08`16      08`16      08`16      08`16      16`08      08`16      08`16
 Hemşire 8      16`08      16`08      08`16      16`08      08`16      08`16      08`16
 Hemşire 9      08`16      08`16      08`16     izinli      08`16      08`16      08`16
Hemşire 10      16`08      08`16      08`16      08`16      08`16      16`08      08`16
Hemşire 11      08`16      08`16      08`16     izinli      16`08      16`08      16`08
Hemşire 12      08`16      16`08      16`08     izinli      08`16      16`08      16`08
Hemşire 13      08`16      08`16      08`16      08`16     izinli      08`16      16`08
Hemşire 14      16`08     izinli      08`16     izinli     izinli      08`16     izinli

Hafta 2:

Hemşireler 08.02.2015 09.02.2015 10.02.2015 11.02.2015 12.02.2015 13.02.2015 14.02.2015
 Hemşire 1      16`08      08`16      16`08      08`16      08`16      08`16     izinli
 Hemşire 2      08`16      08`16      08`16      08`16      08`16     izinli      08`16
 Hemşire 3     izinli      16`08     izinli      08`16      16`08     izinli      16`08
 Hemşire 4      08`16      16`08      08`16      08`16      08`16     izinli      08`16
 Hemşire 5      16`08      08`16      16`08      08`16     izinli     izinli     izinli
 Hemşire 6     izinli      16`08      16`08      08`16      16`08      16`08      08`16
 Hemşire 7      08`16      08`16      08`16     izinli      08`16      16`08     izinli
 Hemşire 8      08`16      08`16      16`08      08`16      16`08      08`16      08`16
 Hemşire 9     izinli      08`16      08`16      08`16      08`16      16`08      08`16
Hemşire 10      16`08      08`16      08`16      16`08     izinli      16`08      08`16
Hemşire 11      08`16     izinli      08`16     izinli      08`16     izinli      16`08
Hemşire 12     izinli      08`16      08`16      08`16     izinli     izinli      08`16
Hemşire 13      16`08      08`16      16`08     izinli      08`16      08`16     izinli
Hemşire 14      08`16      08`16      16`08     izinli      08`16      16`08      08`16

 Hafta 3:

 Hemşireler 15.02.2015 16.02.2015 17.02.2015 18.02.2015 19.02.2015 20.02.2015 21.02.2015
  Hemşire 1     izinli      08`16      08`16      08`16      16`08     izinli     izinli
  Hemşire 2      08`16     izinli      08`16      08`16      08`16     izinli      16`08
  Hemşire 3      08`16      08`16      08`16     izinli      08`16      16`08      08`16
  Hemşire 4      08`16      08`16     izinli      08`16      08`16     izinli     izinli
  Hemşire 5      08`16      08`16      08`16      08`16      08`16      08`16      08`16
  Hemşire 6      08`16      08`16      16`08      08`16      08`16      08`16      16`08
  Hemşire 7      16`08      08`16      16`08      08`16      16`08      08`16      08`16
  Hemşire 8      08`16      08`16     izinli      16`08      08`16     izinli      08`16
  Hemşire 9      08`16      08`16      08`16      16`08      08`16     izinli      08`16
 Hemşire 10      08`16      08`16     izinli      08`16      08`16      08`16      16`08
 Hemşire 11      08`16      16`08      08`16     izinli      08`16      08`16      08`16
 Hemşire 12      08`16      16`08      08`16     izinli     izinli      08`16     izinli
 Hemşire 13      16`08      16`08      08`16      08`16      08`16      08`16      16`08
 Hemşire 14     izinli      08`16     izinli      08`16     izinli      08`16      08`16

Hafta 4:

Hemşireler 22.02.2015 23.02.2015 24.02.2015 25.02.2015 26.02.2015 27.02.2015 28.02.2015
 Hemşire 1      08`16      16`08      16`08     izinli      08`16      16`08      08`16
 Hemşire 2     izinli      08`16      08`16      16`08      08`16      08`16      08`16
 Hemşire 3      08`16      08`16     izinli      08`16      08`16      16`08      08`16
 Hemşire 4      08`16      08`16     izinli      08`16      08`16      08`16      16`08
 Hemşire 5     izinli      16`08      08`16      16`08     izinli     izinli      08`16
 Hemşire 6      08`16      16`08     izinli      08`16      08`16      08`16      08`16
 Hemşire 7      16`08      08`16      08`16      16`08      08`16      16`08      08`16
 Hemşire 8     izinli      08`16      08`16      08`16      08`16      08`16      08`16
 Hemşire 9      08`16     izinli     izinli      16`08     izinli      08`16      16`08
Hemşire 10      08`16      08`16      08`16      08`16      08`16      08`16      08`16
Hemşire 11      08`16     izinli      08`16     izinli      08`16     izinli     izinli
Hemşire 12      08`16      08`16      08`16      08`16      08`16     izinli      08`16
Hemşire 13      16`08      16`08      08`16      16`08      08`16     izinli     izinli
Hemşire 14      08`16     izinli      08`16     izinli      08`16      08`16      16`08
```

Table 2: Şubat Ayı 01.02.2015 ve 07.02.2015 Tarihleri Arası (1 Haftalık)Nöbet Cizelgesi

Şubat Ayı Hafta - 1							
	01.02.2015	02.02.2015	03.02.2015	04.02.2015	05.02.2015	06.02.2015	07.02.2015
Hemşireler	Pazartesi	Salı	Çarşamba	Perşembe	Cuma	Cumartesi	Pazar
Hemşireler 1	izinli	08‘16	izinli	08‘16	08‘16	08‘16	16‘08
Hemşireler 2	izinli	08‘16	16‘08	08‘16	08‘16	08‘16	08‘16
Hemşireler 3	08‘16	16‘08	08‘16	08‘16	16‘08	16‘08	08‘16
Hemşireler 4	08‘16	08‘16	izinli	16‘08	08‘16	izinli	izinli
Hemşireler 5	16‘08	16‘08	izinli	08‘16	08‘16	izinli	08‘16
Hemşireler 6	izinli	16‘08	izinli	08‘16	izinli	08‘16	izinli
Hemşireler 7	08‘16	08‘16	08‘16	08‘16	16‘08	08‘16	08‘16
Hemşireler 8	16‘08	16‘08	08‘16	16‘08	08‘16	08‘16	08‘16
Hemşireler 9	08‘16	08‘16	08‘16	izinli	08‘16	08‘16	08‘16
Hemşireler 10	16‘08	08‘16	08‘16	08‘16	08‘16	16‘08	08‘16
Hemşireler 11	08‘16	08‘16	08‘16	izinli	16‘08	16‘08	16‘08
Hemşireler 12	08‘16	16‘08	16‘08	izinli	08‘16	16‘08	16‘08
Hemşireler 13	08‘16	08‘16	08‘16	08‘16	izinli	08‘16	16‘08
Hemşireler 14	16‘08	izinli	08‘16	izinli	izinli	08‘16	izinli

Table 3: Şubat Ayı 08.02.2015 ve 14.02.2015 Tarihleri Arası (1 Haftalık) Nöbet Cizelgesi

Şubat Ayı Hafta - 2							
	08.02.2015	09.02.2015	10.02.2015	11.02.2015	12.02.2015	13.02.2015	14.02.2015
Hemşireler	Pazartesi	Salı	Çarşamba	Perşembe	Cuma	Cumartesi	Pazar
Hemşireler 1	16‘08	08‘16	16‘08	08‘16	08‘16	08‘16	izinli
Hemşireler 2	08‘16	08‘16	08‘16	08‘16	08‘16	izinli	08‘16
Hemşireler 3	izinli	16‘08	izinl i	08‘16	16‘08	izinli	16‘08
Hemşireler 4	08‘16	16‘08	08‘16	08‘16	08‘16	izinli	08‘16
Hemşireler 5	16‘08	08‘16	16‘08	08‘16	izinli	izinli	izinli
Hemşireler 6	izinli	16‘08	16‘08	08‘16	16‘08	16‘08	08‘16
Hemşireler 7	08‘16	08‘16	08‘16	izinli	08‘16	16‘08	izinli
Hemşireler 8	08‘16	08‘16	16‘08	08‘16	16‘08	08‘16	08‘16
Hemşireler 9	izinli	08‘16	08‘16	08‘16	08‘16	16‘08	08‘16
Hemşireler 10	16‘08	08‘16	08‘16	16‘08	izinli	16‘08	08‘16
Hemşireler 11	08‘16	izinli	08‘16	izinli	08‘16	izinli	16‘08
Hemşireler 12	izinli	08‘16	08‘16	08‘16	izinli	izinli	08‘16
Hemşireler 13	16‘08	08‘16	16‘08	izinli	08‘16	08‘16	izinli
Hemşireler 14	08‘16	08‘16	16‘08	izinli	08‘16	16‘08	08‘16

Table 4: Şubat Ayı 15.02.2015 ve 21.02.2015 Tarihleri Arası (1 Haftalık) Nöbet Cizelgesi

Şubat Ayı Hafta - 3							
	15.02.2015	16.02.2015	17.02.2015	18.02.2015	19.02.2015	20.02.2015	21.02.2015
Hemşireler	Pazartesi	Salı	Çarşamba	Perşembe	Cuma	Cumartesi	Pazar
Hemşireler 1	izinli	08‘16	08‘16	08‘16	16‘08	izinli	izinli
Hemşireler 2	08‘16	izinli	08‘16	08‘16	08‘16	izinli	16‘08
Hemşireler 3	08‘16	08‘16	08‘16	izinli	08‘16	16‘08	08‘16
Hemşireler 4	08‘16	08‘16	izinli	08‘16	08‘16	izinli	izinli
Hemşireler 5	08‘16	08‘16	08‘16	08‘16	08‘16	08‘16	08‘16
Hemşireler 6	08‘16	08‘16	16‘08	08‘16	08‘16	08‘16	16‘08
Hemşireler 7	16‘08	08‘16	16‘08	08‘16	16‘08	08‘16	08‘16
Hemşireler 8	08‘16	08‘16	izinli	16‘08	08‘16	izinli	08‘16
Hemşireler 9	08‘16	08‘16	08‘16	16‘08	08‘16	izinli	08‘16
Hemşireler 10	08‘16	08‘16	izinli	08‘16	08‘16	08‘16	16‘08
Hemşireler 11	08‘16	16‘08	08‘16	izinli	08‘16	08‘16	08‘16
Hemşireler 12	08‘16	16‘08	08‘16	izinli	izinli	08‘16	izinli
Hemşireler 13	16‘08	16‘08	08‘16	08‘16	08‘16	08‘16	16‘08
Hemşireler 14	izinli	08‘16	izinli	08‘16	izinli	08‘16	08‘16

Table 5: Şubat Ayı 22.02.2015 ve 28.02.2015 Tarihleri Arası (1 Haftalık) Nöbet Cizelgesi

Şubat Ayı Hafta - 4							
	22.02.2015	23.02.2015	24.02.2015	25.02.2015	26.02.2015	27.02.2015	28.02.2015
Hemşireler	Pazartesi	Salı	Çarşamba	Perşembe	Cuma	Cumartesi	Pazar
Hemşireler 1	08‘16	16‘08	16‘08	izinli	08‘16	16‘08	08‘16
Hemşireler 2	izinli	08‘16	08‘16	16‘08	08‘16	08‘16	08‘16
Hemşireler 3	08‘16	08‘16	izinli	08‘16	08‘16	16‘08	08‘16
Hemşireler 4	08‘16	08‘16	izinli	08‘16	08‘16	08‘16	16‘08
Hemşireler 5	izinli	16‘08	08‘16	16‘08	izinli	izinli	08‘16
Hemşireler 6	08‘16	16‘08	izinli	08‘16	08‘16	08‘16	08‘16
Hemşireler 7	16‘08	08‘16	08‘16	16‘08	08‘16	16‘08	08‘16
Hemşireler 8	izinli	08‘16	08‘16	08‘16	08‘16	08‘16	08‘16
Hemşireler 9	08‘16	izinli	izinli	16‘08	izinli	08‘16	16‘08
Hemşireler 10	08‘16	08‘16	08‘16	08‘16	08‘16	08‘16	08‘16
Hemşireler 11	08‘16	izinli	08‘16	izinli	08‘16	izinli	izinli
Hemşireler 12	08‘16	08‘16	08‘16	08‘16	08‘16	izinli	08‘16
Hemşireler 13	16‘08	16‘08	08‘16	16‘08	08‘16	izinli	izinli
Hemşireler 14	08‘16	izinli	08‘16	izinli	08‘16	08‘16	16‘08

5 Bulgular ve Tartışma

Sezgisel algoritmalar genellikle en iyiye yakın olan çözüm yoluna hızlı ve kolay şekilde ulaştıklarından, çalışmada sezgisel algoritmalardan biri olan Genetik Algoritma (GA) kullanılmıştır. Genetik Algoritmaların(GA), birden fazla optimal çözüm elde etmesi, çok sayıda parametre ile çalışma imkanı olması ve amaç fonksiyonunu geniş bir açıda araştırması nedeniyle tercih edilmiştir.
Çalışmamızda ilk olarak, çizelgeleme problemlerinde kısıtların çözümünün geçerliliğini belirleyen temel unsurlar olduğu kabul edilmiş ve bu nedenle kısıtları belirlenmişti. Kısıtların hastane yönetiminin amaçlarına, personel isteklerine ve yasal düzenlemelere uygun olmasına özen gösterilmiştir.
Genetik algoritmalar detaylı bir şekilde incelenmiş ve temel kavramlar hakkında bilgiler edinilmiştir. Bu sayede modelimizin doğru adımlar ile oluşturulması sağlanmıştır.
Modelimizin optimal sonuçlar vermesi için tasarımında amaç fonksiyonu, genetik algoritmaların popülasyon yönetimi, çaprazlama, mutasyon gibi bileşenler göz önünde bulundurulmuş ve uygulanmıştır. Ayrıca uygunluk (Fitness) değerlerine göre en optimal çözümlerin elde edilmesi için gerekli işlemler yapılarak ve model tasarlanmıştır.
Çalışmamızda geliştirdiğimiz test modeli üzerinde, test verileri kullanılarak çizelgeleme yapılmış ve uygunluk değerleriyle sonuçlar değerlendirilmiş ve gözlemlennmiştir.
Ardından gerçek veriler ve detaylı kısıtlar dikkate alınarak 28 günlük bir planlama yapılmiş ve elde edilen sonuçlar rapor edilmiştir.
Bu süreç, hem teorik hem de pratik açıdan genetik algoritmaların çizelgeleme problemlerindeki etkinliğini göstermiştir. İleriye dönük çalışmalarda, farklı optimizasyon teknikleri ve daha karmaşık kısıtların nasıl ele alınabileceği üzerinde durulabilir, bu da sağlık hizmetlerinde ve diğer endüstriyel uygulamalarda daha geniş bir kullanım alanı sağlayabilir.

6 Sonuç

Çizelgeleme problemlerinde, insan faktörü sebebiyle hesaplanamayacak çoklukta parametre görünse bile yeterli sayıda matematiksel kısıt yardımıyla gerçek modele çok yakın sonuçlar geliştirilebilir. Büyük boyutlu ve doğrusal olmayan kısıtların olduğu çizelgeleme problemlerinde GA kullanımı da standart çözücülerin kullanıldığı durumdan farksız olup etkin olmayacaktır. Sonuç olarak; farklı sezgisel algoritmaların da kullanılacağı hibrit yaklaşımlar daha etkin çözümler verebilir.

TEŞEKKÜRLER

Başta her daim yanımızda olan, değerli bilgi ve birikimi ile her türlü desteği gösteren Sayın Emre GÜNGÖR hocama ve dönem boyunca her hafta birbirinden değerli sunumlar yaparak kendi konularını üzerinde bizlere farkındalık oluşturmaya çalışan tüm arkadaşlarıma teşekkürlerimi sunarım..

Kaynaça

[1] Ş. İnanç and A. E. Şenaras, "Solving nurse scheduling problem via genetic algorithm in home healthcare," in *Transportation, Logistics, and Supply Chain Management in Home Healthcare: Emerging Research and Opportunities*, pp. 20–28, IGI Global, 2020.

[2] A. Küçük and İ. D. KOCAKOÇ, "Hemşire çizelgeleme problemlerinin genetik algoritmalarla optimizasyonu ve bir uygulama," *Manisa Celal Bayar Üniversitesi Sosyal Bilimler Dergisi*, vol. 19, no. Armağan Sayısı, pp. 203–210, 2021.

[3] M. Yamamura, S. Kobayash, M. Yamagishi, and H. Ase, "Nurse scheduling by genetic algorithms," *Hiroaki Kitano: Genetic Algorithms*, vol. 2, pp. 89–125, 1993.

[4] R. Bailey, K. Garner, and M. Hobbs, "Using simulated annealing and genetic algorithms to solve staff-scheduling problems," *Asia-Pacific Journal of Operational Research*, vol. 14, no. 2, p. 27, 1997.

[5] U. Aickelin and K. A. Dowsland, "An indirect genetic algorithm for a nurse-scheduling problem," *Computers & operations research*, vol. 31, no. 5, pp. 761–778, 2004.

[6] A. Duenas, G. Y. Tütüncü, and J. B. Chilcott, "A genetic algorithm approach to the nurse scheduling problem with fuzzy preferences," *IMA Journal of Management Mathematics*, vol. 20, no. 4, pp. 369–383, 2009.

[7] S. S. Balekar and N. Mhetre, "Survey of genetic algorithm approach for nurse scheduling problem," *International Journal of Science and Research*, vol. 4, no. 6, pp. 55–62, 2013.

[8] K. Leksakul, S. Phetsawat, *et al.*, "Nurse scheduling using genetic algorithm," *Mathematical Problems in Engineering*, vol. 2014, 2014.

[9] J. Kim, W. Jeon, Y.-W. Ko, S. Uhmn, and D.-H. Kim, "Genetic local search for nurse scheduling problem," *Advanced Science Letters*, vol. 24, no. 1, pp. 608–612, 2018.

[10] N. Alfadilla, P. Sentia, D. Asmadi, *et al.*, "Optimization of nurse scheduling problem using genetic algorithm: a case study," in *IOP Conference Series: Materials Science and Engineering*, vol. 536, p. 012131, IOP Publishing, 2019.

[11] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[12] A. Elen, "Çizelgeleme probleminin sezgisel optimizasyon yaklaşımıyla çözümü," Master's thesis, Fen Bilimleri Enstitüsü, 2011.

[13] S. Biroğul, "Genetik algoritma yaklaşımıyla atölye çizelgeleme," *Gazi Üniversitesi Fen Bilimleri Enstitüsü Yüksek Lisans Tezi, Gazi Üniversitesi Fen Bilimleri Enstitüsü, ANKARA, Ocak*, 2005.

[14] E. Aydemir, "Atölye tipi çizelgeleme problemlerinin öncelik kuralı tabanlı genetik algoritma yaklaşımıyla simülasyon destekli optimizasyonu," Master's thesis, Fen Bilimleri Enstitüsü, 2009.

[15] miracöztürk, "Business intelligence specialist." `https://miracozturk.com/python-ile-siniflandirma-analizleri-genetik-algoritmalar/`. Erişim Tarihi: 15 Nisan 2024.

[16] Tobs80, "Schedule with genetic algorithms main." `https://github.com/Tobs80/Schedule-with-Genetic-Algorithms/blob/main/main.py`. Erişim Tarihi: 01 Mayıs 2024.