Genetik Algoritma ile Çift Taraflı Montaj Hattı Dengeleme

Bülent SİYAH

İçindekiler

1. GİRİŞ

Bu çalışmada, çift taraflı mondaj hattının her iki tarafı için belirlenmiş çevrim süresi içerisinde istasyon sayısını en aza indirgeme amaçlanmıştır. Operasyonlarda taraf ve istasyon sayısı kısıtları bulunmaktadır. Problemin Genetik Algortima ile çözümü için geliştirilen uygulama, c#(sharp) programlama dili ile yazılmıştır.

Genetik algoritma rastgele arama metodu olduğu için tek bir çözüm aramak yerine bir çözüm kümesi üzerinde çalışır. Optimum çözüme olası çözümlerin bir bölümü üzerinde gidilir. Böylece çalışmadaki sonuçlar her zaman en iyi olmaz. Çalışmada genetik algoritmanın kullanılmasının nedeni, genetik algoritmanın problemin doğasıyla ilgili herhangi bir bilgiye ihtiyaç duymamasıdır.

2. Materyal ve Yöntem

Genetik Algoritmalar, daha iyi çözümlere yavaş yavaş bir yaklaşım sağlayan geniş bir problem uzayı boyunca yönlendirilmiş rastgele bir araştırmaya imkan sağlar. Temel avantajı optimize edilmeye çalışılan problemin doğasıyla ilgili herhangi bir bilgiye ihtiyaç göstermemeleridir.

2.1. Genetik Algoritmaların Temel Kavramları

Evrimsel hesaplama topluluğunda yerleşik, tek bir Genetik Algoritma tanımı bulunmamakla beraber, Genetik Algoritma sınıfına girdiği kabul edilen yöntemlerde en azından şu ortak unsurlara rastlanır: kromozom topluluğu, uygunluğa göre seçilim, yeni bireyler üretmek için çaprazlama, yeni bireylerin rastgele mutasyonu. Holland'ın GA'nın unsurları içinde saydığı ters çevirme operatörü günümüzde nadiren kullanılmaktadır.

2.1.1. Kromozom ve Topluluk

GA'da kromozomlar, problem için olası çözümleri temsil ederler. Bu çözümlerden her birine birey adı verilir. Topluluk (popülasyon) kromozomlardan (bireylerden) oluşan kümedir. GA'nın kromozom toplulukları üzerinde işlemler yapması ile, bir önceki topluluklar her seferinde yeni üretilen topluluklarla yer değiştirir.

2.1.2. Uygunluk

Uygunluk değeri, çözümün kalitesini belirler ve uygunluk fonksiyonu kullanılarak hesaplanır. Uygunluk fonksiyonu mevcut topluluktaki her kromozoma bir puan (uygunluk değeri) atar. Kromozomun uygunluğu, o kromozomun eldeki problemi ne kadar iyi çözdüğüyle ilişkilidir.

2.1.3. Genetik Operatörler

GA'nın en basit formu üç tip genetik operatör içerir: seçilim, çaprazlama (çift noktalı) ve mutasyon.
Seçilim: Bu operatör topluluktaki kromozomları üreme için seçer. Kromozomun yüksek uygunluk değerine sahip olması, üremek için seçilmesi ihtimalini arttırır.
Çaprazlama: Bu operatör rastgele iki lopus belirleyip, iki kromozomun bu lopuslardan önceki ve sonraki kısımlarını aynı kalıcak şeklide, lopus aralarındaki kısımları değiştirilerek gerçekleştirilir.
Mutasyon: Bu operatör kromozomdaki bir veya daha fazla rastgele alınan bireyin yine rastgele seçilen gen ile genin değerine sahip diğer genin değişimi ile olur.

2.2. Genetik Algoritma Parametreleri

2.2.1. Topluluk Büyüklüğü

Topluluk büyüklüğü için seçilen değer, algoritmanın performansını iki şekilde etkilemektedir. Birincisi, topluluk büyüklüğünün aşırı küçülmesi araştırma uzayının yetersiz örneklenmesine sebep olacağından kontrollü ıraksamayı sağlamak zorlaşacak ve araştırma belirli bir alt optimal noktaya doğru sürüklenecektir. ikincisi, topluluk için aşırı yüksek değer seçildiğinde bir nesillik gelişim oldukça uzun süreye ihtiyaç duymaktadır.

2.2.2. Çaprazlama Oranı

Bireylerin çoğalması sırasında kromozomlara uygulanacak çaprazlama operatörünün frekansını belirlemek amacıyla kullanılan parametredir. Düşük çaprazlama oranı yeni nesle çok az sayıda yeni yapının (ebeveyninden farklı yeni bireyler) girmesine sebep olmaktadır. Dolayısıyla tekrar üreme operatörü algoritmada aşırı etkili bir operatör haline gelmekte ve araştırmanın yakınsama hızı düşmektedir. Yüksek çaprazlama oranı araştırma uzayının çok hızlı bir şekilde araştırılmasına sebep olmaktadır. Ama oran aşırı yüksek ise, çaprazlama operatörü benzer veya daha iyi yapıları üretmeden kuvvetli olan yapılar çok hızlı olarak bozulduğundan, algoritmanın performansı düşmektedir.

2.2.3. Mutasyon Oranı

Mutasyon operasyonun frekansı, etkili bir genetik algoritma tasarlamak için çok iyi kontrol edilmelidir. Mutasyon operasyonu, araştırma sahasına yeni bölgelerin girmesini sağlar. Yüksek mutasyon oranı, araştırmaya aşırı bir rastgelelik kazandıracak ve araştırmayı çok hızlı olarak ıraksatacaktır. Başka bir deyişle, topluluğun gelişmesine değil tahribatına sebep olacaktır. Bu durumun tersine, çok düşük mutasyon oranının kullanılması ıraksamayı aşırı düşürecek ve araştırma uzayının tamamen araştırılmasını engelleyecektir. Dolayısıyla, algoritmanın alt optimal çözüm bulmasına sebep olacaktır.

2.3. Genetik Algoritma Süreci

Çözülmek üzere tanımlı bir problem verilmesi ve aday çözümlerin birer sayı dizisi ile temsil edilmesi halinde, GA şu şekilde çalışır:

1. N adet kromozom (problem için aday çözümler) içeren rastgele oluşturulmuş bir topluluk ile başla.
2. Topluluktaki her x kromozomu için f(x) uygunluk değerini hesapla.
3. Aşağıdaki adımları birey n (topluluk büyüklüğü) oluşturuluncaya kadar tekrarla.

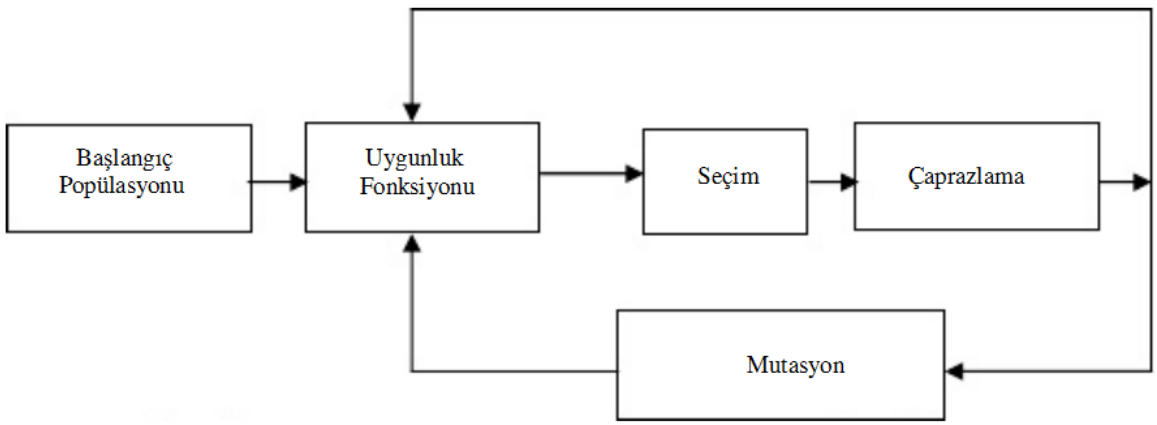
3.a. Güncel topluluktan, yüksek uygunluk değerinin seçilme ihtimalini arttırdığını göz önünde bulundurarak, iki ebeveyn kromozom seç. Seçilim, aynı kromozomun birden çok defa ebeveyn olarak seçilmesine olanak verecek şekilde yapılır.

3.b. Pc olasılığı ("çaprazlama olasılığı" ya da "çaprazlama oranı") ile seçilen çifti iki yeni birey oluşturmak üzere rastgele belirlenen iki noktadan çaprazla. Eğer çaprazlama gerçekleşmezse ebeveynlerinin birebir kopyası olan iki çocuk oluştur. Pc için iki noktalı çaprazlama yöntemi seçilmiştir.

3.c. Pm olasılığı ("mutasyon olasılığı" ya da "mutasyon oranı") ile, oluşan bir veya birden fazla çocuğu tüm veya bazı lopuslarında mutasyona uğrat. Lopuslardaki genlerin değiştirilmesi ile de mutasyon gerçekleştirilir.

3.d. Sonuçta elde edilen kromozomları yeni topluluğa ekle.

4. Önceki topluluğu yeni topluluk ile değiştir. Böylece yeni nesil elde edilmiş oldu.
5. Sonlandırma koşulu sağlandıysa mevcut topluluktaki en iyi çözümü döndür, sağlanmadıysa 2. adıma dön. Sonlandırma koşulu çoğu zaman nesil sayısıdır. Bazı genetik algortimalarda bu koşul, belirlenen aralıkta uygunluk değerine sahip birey elde edilmesi olabilir.



Şekil 2.1.Genetik Algoritmalarda evrimleşme döngüleri

2.4. Sürecin Probleme Uygulanışı

2.4.1. İlk Toplum Oluşturulması ve Kromozomların Kodlanması

Bir GA programının yazılması işleminde karşılaşılan ilk problemlerden biri kromozomların kodlanmasıdır. Kodlama işlemi problemin türüne göre değişmektedir. Bazen binary (ikili) kodlama en iyi çözüm bulmada kullanılırken, bazen de permutasyon veya değer kodlama yöntemleri optimum sonucu bulmada en uygun kodlama yöntemi olabilir. Permutasyon kodlama yöntemi, genel olarak gezgin satıcı ve şebeke tasarımları gibi sıra gözeten problemlerde yaygınca kullanılır.

Kromozom	**A**	**3 5 1 2 7 6 0 4**
Kromozom	**B**	**0 1 5 6 2 3 4 7**

Şekil 2.2.Permutasyon kodlama için kromozom örneği

Problemin çözümü için populasyon büyüklüğü karar verilmeli, populasyon büyüklüğü seçilirken aşırı yüksek seçilirse gelişim yavaşlar, aşırı küçük seçilmesi durumunda da araştırma uzayı yetersiz olucağından problemimize en uygun populasyon büyüklüğü 30 birey olarak seçilmiştir. Programın arayüzünde birey sayısı değiştirilebilir. Problemde permutasyon kodlama yöntemi seçilip, hatların taraflarındaki operasyon sayıları farklı olduğundan, hattın sol tarafı 10 operasyon dolayısıyla 10 gen ile, hattın sağ tarafı 17 operasyon yani 17 gen ile temsil edilmiştir. Her gendeki sayı değeri operasyon numarasına eşittir. Örnek olarak hattın sol tarafındaki 7 nolu kromozomun genleri: 4051296741038 şeklinde kodlanmıştır.

İlk toplum oluşturulurken tüm bireylerin genleri rastgele oluşturulmuştur. Problemde ilk toplumu oluşturan fonksiyon sadece ilk nesilde çalışır. Diğer nesillerde zaten eski ve yeni nesil değişimi gerçekleştiğinden bu fonksiyon kullanılmaz. Fonksiyon, birey sayısı kadar dönen döngüde her birey için gerekli gen sayısı kadar rastgele sayı üretir. Permutasyon kodlama ile yapılırken aynı gen değerine sahip bireylerin oluşması engellenmiştir.

- İlk toplumun oluşturulduğu ve kromozomların rastgele kodlandığı kod bloğu:

```
public void ilkToplumOlustur()
{
    ArrayList randomSayilar = new ArrayList();
    for (int iii = 1; iii <= BireySayisi; iii++)
    {
        randomSayilar.Clear();
        for (int jjj = 1; jjj <= GenSayisi; jjj++)
        {
            int randomSayi = Sayi.Next(1, GenSayisi + 1);
            if (randomSayilar.IndexOf(randomSayi) == -1)
            {
                ilkToplum[iii, jjj] = randomSayi;
                randomSayilar.Add(randomSayi);
            }
            else
            {
                jjj = jjj - 1;
            }
        }
    }
}
```

2.4.2. Toplumdaki Her Kromozomun Uygunluk Değerinin Ölçülmesi

Popülasyon içerisinde yer alan ve her bir bireyi temsil eden kromozomların ne kadar iyi olduklarını bulan fonksiyona uygunluk fonksiyonu (uyumluluk, fitness) fonksiyonu denir. Bu GA programında özel çalışan tek kısımdır. Uygunluk fonksiyonu kromozomların şifrelerini çözer (decoding) ve sonra da hesaplama yaparak bu kromozomların uygunluklarını hesaplar.Elde edilen uygunluk değerlerine göre kromozomların yeni generasyon da olup olmayacaklarına karar verilir. Bütün popülasyonun (topluluğun) uygunluk değerleribelirlendiğinde sonuçlandırma kriterinin sağlanıp sağlanmadığına bakılır. Sonuçlandırma kriteri birkaç şekilde seçilebilmektedir. Bu seçeneklerden biri, istenilen nesil sayısına ulaşılması olabileceği gibi, tüm topluluğun uygunluk değerinin en iyi çözümün uyumluluk değerine oranının yeterli görülen bir değere ulaşması olabilir.

Uygunluk değerleri şöyle hesaplanır. Genler operasyonları temsil edip, sıra ile gruplandırılarak işçinin net çalışma süresi(260 dk.) ile işlemlerden geçirilip, kayıp zaman bulunarak hesaplanır. Buradaki gruplam için şart belli bir eşik değere göre yapılır. Eşik değeri sağlayan operasyonlar yani genler aynı grupta yer alarak o istasyona atanır. İşçinin çalışma süresi ile işlemler şunlardır. Eşik değeri sağlayan gen grubunun tüm süreleri toplanır. Bu toplam değer işçinin çalışma süresine göre modu alınır. Alınan mod ile elde edilen değer kalan değeridir. Toplam değer yine işçinin çalışma süresine göre normal bölme işleminden geçirilir. İşlem sonucunda elde edilen değer bölüm değerdir. Uygunluk değeri için ilk olarak işçinin çalışma süresi ile elde edilen kalan değer çıkartılır. Bu değer, elde edilen bölüm süresinin bir fazlası ile çarpılarak uygunluk değeri hesaplanır. İşlemlerde şu ayrıntı önemlidir. Eşik değeri geçememe ihtimali olursa tüm operasyonlar tek istasyonda toplanır. Bu istenmeyen durum için kayıp zaman, tüm operasyonların toplamı ile elde edilir. Böylece problemde bu adımda tekli gruplamanın uygunluk değeri yüksek olduğundan elenme ihtimali çok yükselir. Uygunluk fonksiyonunda eşik değer istenilirse her iki hattın tarafı için ayrı değerlere sahip olabilir. Fonksiyon sonrasında tüm bireylerin uygunluk değerleri bir dizide toplanır. Dizilerin birbirleriyle indexleri sayesinde anlaşırlar.

- Toplumdaki her kromozomun uygunluk değerinin hesaplandığı kod bloğu:

```
        public void Uygunluk()
        {
            ArrayList UygunlukistasyonDizileri = new ArrayList();
            int deger = 0;
            int esik;
            if (GenSayisi > SabitGenSayisi)
            {
                esik = SagEsikDegeri;
            }
            else
            {
                esik = SolEsikDegeri;
            }
            for (int k = 1; k <= BireySayisi; k++)
            {
                UygunlukistasyonDizileri.Clear();
                ilkToplumBireyUygunlugu[k] = 0;
                for (int l = 1; l <= GenSayisi; l++)
                {
                    UygunlukistasyonDizileri.Add(ilkToplum[k, l]);
                    deger = UygunlukDegerleri(UygunlukistasyonDizileri);
                    if (deger <= esik || l == GenSayisi)
                    {
                        ilkToplumBireyUygunlugu[k] = ilkToplumBireyUygunlugu[k] +
deger;
                        UygunlukistasyonDizileri.Clear();
                    }
                }
            }
        }
        public int UygunlukDegerleri(ArrayList GelenDiziKumesi)
        {
            int degertopla = 0;
            int degertoplakalan = 0;
            int degertoplabolum = 0;
```

```
            int uygunlukDegeri = 0;
            for (int kkk = 0; kkk < GelenDiziKumesi.Count; kkk++)
            {
                degertopla = degertopla +
GenlerinSureleri[Convert.ToInt32(GelenDiziKumesi[kkk])];
            }
            degertoplakalan = degertopla % OrtalamaCalismaSuresi;
            degertoplabolum = degertopla / OrtalamaCalismaSuresi;
            uygunlukDegeri = OrtalamaCalismaSuresi - degertoplakalan;
            uygunlukDegeri = uygunlukDegeri * (degertoplabolum + 1);
            if (GelenDiziKumesi.Count == GenSayisi)
            {
                uygunlukDegeri = degertopla;
            }
            return uygunlukDegeri;
        }
```

2.4.3. Yeni Toplum Oluştur

2.4.3.1. Elitizm Yapılması

Elitizm, performansı en iyi bireyle (bireyler) temsil edilen o ana kadar ki en iyi çözümün korunarak, değiştirilmeksizin bir sonraki nesle aktarılmasını sağlar. Toplumda tüm bireyler çaprazlanarak oluşturulursa, iyi olan bireylerin yok olma veya çözümden uzaklaşma ihtimali ortaya çıkar. Bu durumu ortadan kaldırmak için toplum içindeki en iyi 2 tane birey direk yeni topluma eklenerek elitizm yapıldı. Elitizm yapıldığı fonksiyonda en iyi uygunluk değerine sahip bireylerin seçilmesi dışında tüm bireyler uygunluk değerlerine göre sıralandı. Böylece çarpazlama için seçilen bireylerin kıyaslama işlemleri daha kolay gerçekleştirildi.

- Elitizmin yapıldığı kod bloğu:

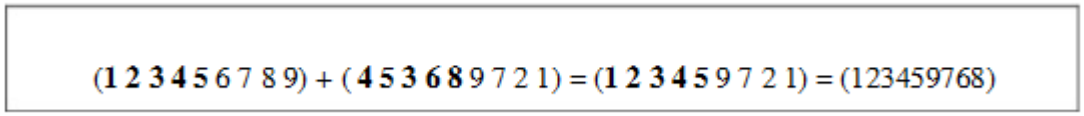
```
        public void Elitizm()
        {
            int EnKucuk, EnKucukindex;
            for (int elit = 1; elit <= BireySayisi; elit++)
            {
                EnKucuk = 100000;
                EnKucukindex = 0;
                int gecici = 0, gecici2 = 0;
                for (int elit2 = elit; elit2 <= BireySayisi; elit2++)
                {
                    if (ilkToplumBireyUygunlugu[elit2] < (EnKucuk))
                    {
                        EnKucuk = ilkToplumBireyUygunlugu[elit2];
                        EnKucukindex = elit2;
                    }
                }
                gecici = ilkToplumBireyUygunlugu[elit];
                ilkToplumBireyUygunlugu[elit] = ilkToplumBireyUygunlugu[EnKucukindex];
                ilkToplumBireyUygunlugu[EnKucukindex] = gecici;
                for (int degisenGenler = 1; degisenGenler <= GenSayisi;
degisenGenler++)
                {
                    gecici2 = ilkToplum[elit, degisenGenler];
                    ilkToplum[elit, degisenGenler] = ilkToplum[EnKucukindex,
degisenGenler];
                    ilkToplum[EnKucukindex, degisenGenler] = gecici2;
                }
            }
            for (int index = 1; index <= ElitSayisi; index++)
            {
                for (int bb = 1; bb <= GenSayisi; bb++)
                {
                    YeniToplum[index, bb] = ilkToplum[index, bb];
```

```
            }
        }
    }
```

2.4.3.2. Seçilim ve Çaprazlama

En iyi kromozomları seçmek için birçok yöntem vardır. Bu seçim yöntemlerinden bazıları; Rulet tekerleği seçimi (roulette wheel selection), Boltzman seçimi (Boltzman selection), turnuva seçimi (tournament selection), sıralama seçimi (rank selection) ve sabit durum seçimidir. Seçilimde ikili turnuva seleksiyonu kullanıldı. İkili turnuva seçiminde populasyon içinden rastgele iki birey seçilir ve uygunluklarına göre iyi olan alınır, daha sonra tekrar rastgele iki birey seçilir ve yine uygunluklarına göre en iyi olan alınır. Böylece finale kalan iki tane birey çaprazlama işlemi için seçilmiş olur.

Genel olarak çaprazlama işlemi; kromozomların nasıl kodlanacağı belirlendikten sonra popülasyondaki kromozomlardan uygunluk değeri yüksek olanlar içerisinden rasgele seçilen iki birey arasında, yine rasgele seçilen bir noktaya göre karşılıklı gen alış-verişi olarak tanımlanır. Çaprazlama işlemi, uygunluk değeri iyi olan ve rasgele seçilen iki kromozomdan daha iyi uygunluk değerine sahip, yeni kromozomların elde edilmesi amacıyla yapılmaktadır. Yeni neslin uygunluk değerlerinin iyi çıkması ve problemin optimum çözüme ulaşabilmesi açısından çaprazlama işlemi son derece önem taşımaktadır. Bu aşamada, faklı kromozomlar alınarak çaprazlama işlemi yapıldığı için değişik özelliklerin test edilmesine ve hızlı olarak yeniden oluşmasına olanak sağlar. Bu işlem neticesinde ise, iki bireyin kopyası durumundaki kopyalanmış bireyler ile çeşitlilik sağlanmış ve en iyi çözüme yaklaşım sağlanmıştır. Çaprazlama için uygulamada iki noktalı çaprazlama seçilmiştir. Bu tür çaprazlama işleminde iki tane kırılma noktası alınır. İlk noktaya kadar olan bitler birinci kromozomdan, iki nokta arasındaki bitler ikinci kromozomdan, kalanlar ise tekrar birinci kromozomdan yeni bireye kopyalanırlar.



Şekil 2.3. Permutasyon kodlamada çaprazlama

Uygulamada seçilen ikili turnuva metodunu uygulamak için toplumda rastgele 2 şer birey alınıp birbirleriyle kıyaslanır. Bu kıyaslama işlemini, daha önce elitizm fonksiyonunda yapılmış olan bireylerin uygunluklarına göre sıralanmış olması kolaylaştırır. Çünkü rastgele elde edilen sayıların, sayı değerlerini kıyaslamak yeterli olacaktır. Çaprazlama için seçilen iki bireyler iki noktalı yöntem ile çaprazlanıp yeni topluma aktarılır.

- Seçilim ve çaprazlamanın yapıldığı kod bloğu:

```
        public void Caprazlama()
        {
            int galip1, galip2, rastgeleBirey1, rastgeleBirey2, rastgeleBirey3,
rastgeleBirey4, lopus1, lopus2, gecicilopus;
            for (int carpazi = ElitSayisi + 1; carpazi <= BireySayisi; carpazi =
carpazi + 2)
            {
                if (carpazi == BireySayisi)
                {
                    carpazi--;
                }
                rastgeleBirey1 = Sayi.Next(1, BireySayisi);
                rastgeleBirey2 = Sayi.Next(1, BireySayisi);
                rastgeleBirey3 = Sayi.Next(1, BireySayisi);
                rastgeleBirey4 = Sayi.Next(1, BireySayisi);
                lopus1 = Sayi.Next(2, GenSayisi - 1);
                lopus2 = Sayi.Next(2, GenSayisi - 1);
                if (lopus1 > lopus2)
                {
                    gecicilopus = lopus1;
                    lopus1 = lopus2;
                    lopus2 = gecicilopus;
```

```
                }
                if (rastgeleBirey1 > rastgeleBirey2)
                {
                    galip1 = rastgeleBirey2;
                }
                else
                {
                    galip1 = rastgeleBirey1;
                }
                if (rastgeleBirey3 > rastgeleBirey4)
                {
                    galip2 = rastgeleBirey4;
                }
                else
                {
                    galip2 = rastgeleBirey3;
                }
                for (int b = 1; b < lopus1; b++)
                {
                    YeniToplum[carpazi, b] = ilkToplum[galip1, b];
                    YeniToplum[carpazi + 1, b] = ilkToplum[galip2, b];
                }
                for (int b = (lopus1); b < lopus2; b++)
                {
                    YeniToplum[carpazi, b] = ilkToplum[galip2, b];
                    YeniToplum[carpazi + 1, b] = ilkToplum[galip1, b];
                }
                for (int b = (lopus2); b <= GenSayisi; b++)
                {
                    YeniToplum[carpazi, b] = ilkToplum[galip1, b];
                    YeniToplum[carpazi + 1, b] = ilkToplum[galip2, b];
                }
                for (int hataliBirey = carpazi; hataliBirey <= carpazi + 1;
hataliBirey++)
                {
                    ArrayList HataliGenDuzelt = new ArrayList();
                    for (int hataliGen = 1; hataliGen <= GenSayisi; hataliGen++)
                    {
                        if (HataliGenDuzelt.IndexOf(YeniToplum[hataliBirey,
hataliGen]) == -1)
                        {
                            HataliGenDuzelt.Add(YeniToplum[hataliBirey, hataliGen]);
                        }
                        else
                        {
                            HataliGenDuzelt.Add("*");
                        }

                    }
                    while (HataliGenDuzelt.Contains("*"))
                    {
                        for (int EksikGen = 1; EksikGen <= GenSayisi; EksikGen++)
                        {
                            int hataGen = HataliGenDuzelt.IndexOf("*");
                            if (HataliGenDuzelt.IndexOf(EksikGen) == -1)
                            {
                                HataliGenDuzelt.RemoveAt(hataGen);
                                HataliGenDuzelt.Insert(hataGen, EksikGen);
                            }
                        }
                    }
```

```
                    for (int arrayiGeneAta = 1; arrayiGeneAta <= GenSayisi;
arrayiGeneAta++)
                    {
                        YeniToplum[hataliBirey, arrayiGeneAta] =
Convert.ToInt32(HataliGenDuzelt[arrayiGeneAta - 1]);
                    }

                }
            }
        }
```

2.4.3.3. Mutasyon

GA programının sonuca ulaşmasında önemli olan başka bir etkende mutasyon işlemidir. Programın belirli bir alana yoğunlaşarak sonuca ulaşmayı engelleyen bazı durumlarda, problemin daha geniş bir alanda arama yapabilmesi için mutasyon işlemine baş vurulur. Böylelikle popülasyondaki çözümlerin yerel optimuma düşmesi engellenmiş olur. Uygulamada çaprazlamaya uğrayan bireylerden biri mutasyona uğratıldı. Mutasyon için rastgele bir birey seçilir ve yine rastgele seçilen gen ile genin değerine sahip olan diğer gen yerdeğiştirilerek gerçekleştirilir.

- Mutasyonun yapıldığı kod bloğu:

```
        public void Mutasyon()
        {
            int MutandBirey, mutgen1, mutgen2;
            String sabitgen1, sabitgen2;
            for (int mutand = 1; mutand <= MutasyonSayisi; mutand++)
            {
                MutandBirey = Sayi.Next(1, BireySayisi + 1);
                mutgen1 = Sayi.Next(0, GenSayisi);
                mutgen2 = Sayi.Next(0, GenSayisi);
                ArrayList MutasyonDuzelt = new ArrayList();
                for (int MutasGen = 1; MutasGen <= GenSayisi; MutasGen++)
                {
                    MutasyonDuzelt.Add(YeniToplum[MutandBirey, MutasGen]);
                }
                sabitgen1 = MutasyonDuzelt[mutgen1].ToString();
                sabitgen2 = MutasyonDuzelt[mutgen2].ToString();
                MutasyonDuzelt.RemoveAt(mutgen1);
                MutasyonDuzelt.Insert(mutgen1, sabitgen2);
                MutasyonDuzelt.RemoveAt(mutgen2);
                MutasyonDuzelt.Insert(mutgen2, sabitgen1);
                if (mutgen1 == mutgen2)
                {
                    mutand--;
                }
                for (int MutanGenAta = 1; MutanGenAta <= GenSayisi; MutanGenAta++)
                {
                    YeniToplum[MutandBirey, MutanGenAta] =
Convert.ToInt32(MutasyonDuzelt[MutanGenAta - 1]);
                }
            }
        }
```

2.4.4. Önceki Toplum ile Yeni Toplumu Değiştir

Önceki topluluğu yeni topluluk ile değiştirilir. Böylece yeni nesil elde edilmiş olur.

- Eski toplum ile yeni toplumun yerdeğiştirdiği kod bloğu:

```
        public void ToplumDegistir()
        {
            for (int toplumDegistirBirey = 1; toplumDegistirBirey <= BireySayisi;
toplumDegistirBirey++)
```

```
            {
                for (int toplumDegistirGen = 1; toplumDegistirGen <= GenSayisi;
toplumDegistirGen++)
                {
                    ilkToplum[toplumDegistirBirey, toplumDegistirGen] =
YeniToplum[toplumDegistirBirey, toplumDegistirGen];

                }
            }
        }
```

2.4.5. Sonlandırma Koşulu Kontrolu

Problemin çözümü için sonlandırma koşulu belirelenen iterasyon sayısı kabul edilir. Döngü sonlanınca problemin en uygun çözümü elde edilmiş olur.

- Sonlandırma koşulu yani nesil sayısı kadar döngünün bulunduğu kod bloğu:

```
public void Genetik()
{
    ilkToplumOlustur();
    for (int ii = 1; ii <= NesilSayisi; ii++)
    {
        Uygunluk();
        Elitizm();
        Caprazlama();
        Mutasyon();
        ToplumDegistir();
        if (ii == NesilSayisi)
        {
            SonUygunluk();
            EkranCiz();
        }
    }
}
```

3. PERFORMANS ANALİZİ

Çalışmada, çift taraflı mondaj hattının her iki tarafı için belirlenmiş çevrim süresi içerisinde istasyon sayısını en aza indirgeme hedeflenmiştir. Problemde her hattın belirli bir operasyon süresi vardır. Bu operasyonların bir araya geldiği istasyonlarda işlemleri gerçekleştirmek için operasyonların toplamı kadar süre harcanır. Bu süre bir işçinin çalışma süresinin tam katları şeklinde yerleştirilmelidir. Aksi durumda operasyonarın bitiminde işçinin bekleme yani kayıp süre oluşur. Bu uygulamada da amaç genetik algoritma ile bu süreyi olabildiğince en aza indirmek hedeflenmiştir.

3.1. Operasyonlar ve Çevrim Süreleri

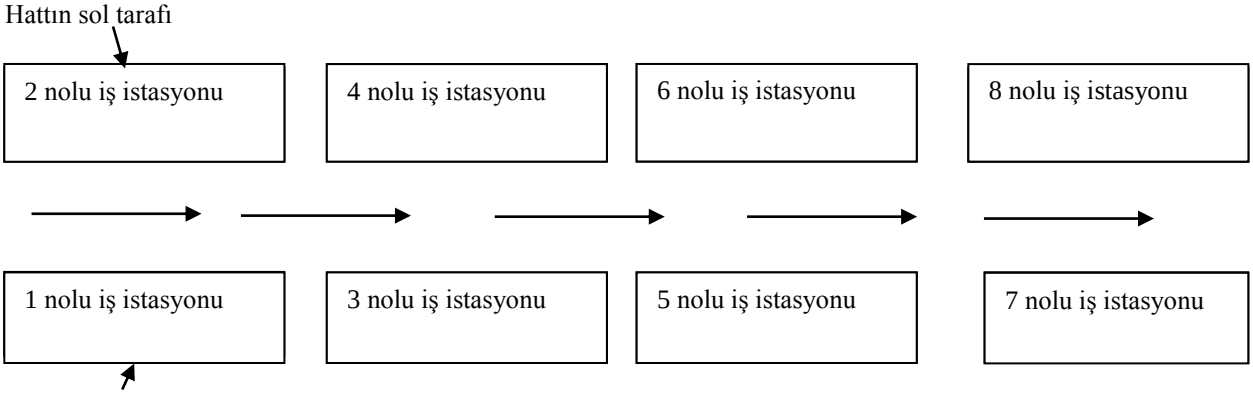
Problemin tanımlanması için her istasyon için değişmez ve ayrı olan operasyon numaraları ile bu operasyonlara ait süreler tablo şeklinde verilmiştir.

Operasyon Numarası	Hattın Sol Tarafındaki Operasyonlar (dk.)	Hattın Sag Tarafındaki Operasyonlar (dk.)
1		97
2		59
3		124
4		72
5		58

6		42
7	**124**	
8	**79**	
9	**74**	
10		120
11		150
12		181
13	**85**	
14	**41**	
15		126
16		70
17		116
18		91
19		114
20		79
21	**45**	
22	**91**	
23	**189**	
24	**72**	
25	**121**	
26		134
27		28

3.2. Problemin Varolan Durumu

İstasyonlar için şuan ki durum :

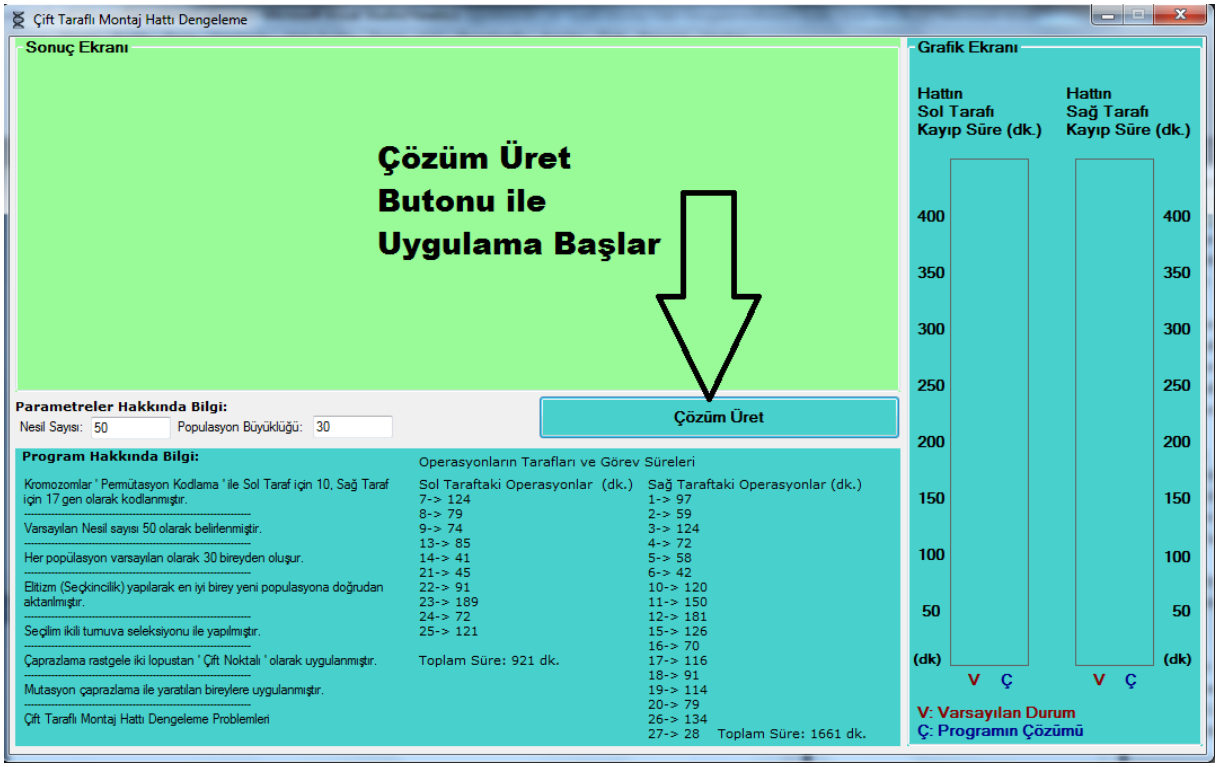


İstasyonlar No	**İstasyona Atanan Operasyonlar**	**Operasyonların Toplam Süresi (dk.)**	**Kayıp Süre (dk.)**
1	1,2,3,4,5,6	452	67
2	**7,8,9**	**277**	**222**
3	10,11,12	451	68
4	**13,14**	**126**	**134**
5	15,16,17,18,19,20	596	183
6	**21,22,23,24,25**	**518**	**2**
7	26,27	162	318
8	---------	**0**	**0**

	Hattın sol tarafı (dk.)	**Hattın sağ tarafı (dk.)**
Toplam Kayıp Süre:	358	416

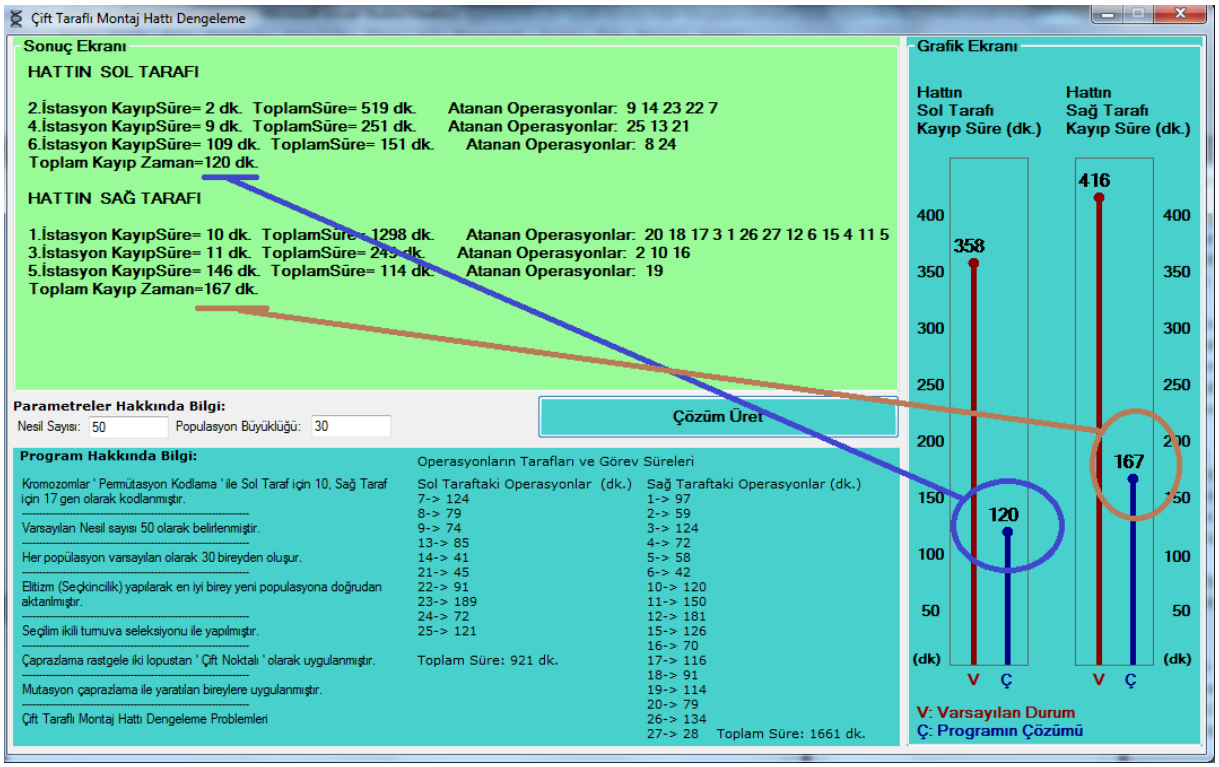
3.3. Uygulamanın Arayüzü

Problemin çözümünü için geliştirilen uygulamanın arayüzünde nesil ve birey sayıları değiştirilebilir, program başlatılır.



Şekil 3.1. Uygulamanın arayüz görünümü

Şekil 3.2. Arayüz görünümü (Nesil ve Birey Sayısının Değiştirilebilmesi)

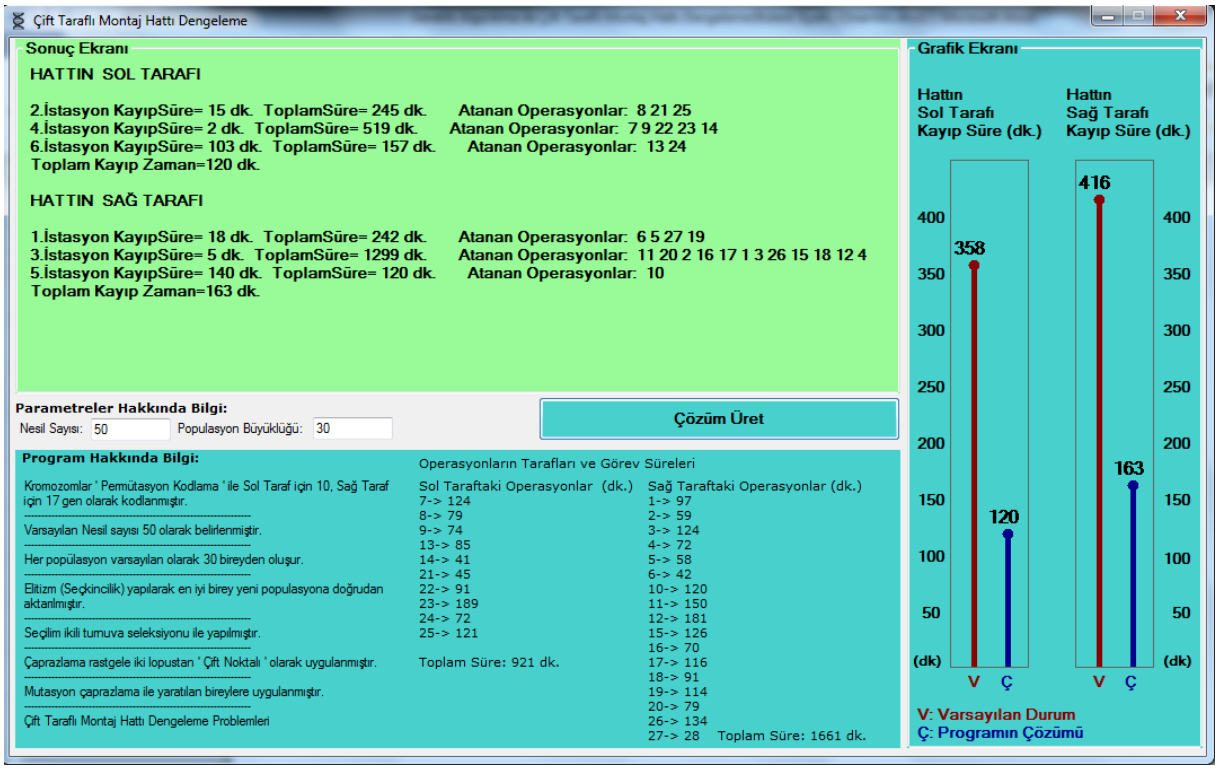


Şekil 3.3. Sonuçların grafik ekranında karşılıkları

3.4. Uygulama Sonuçları

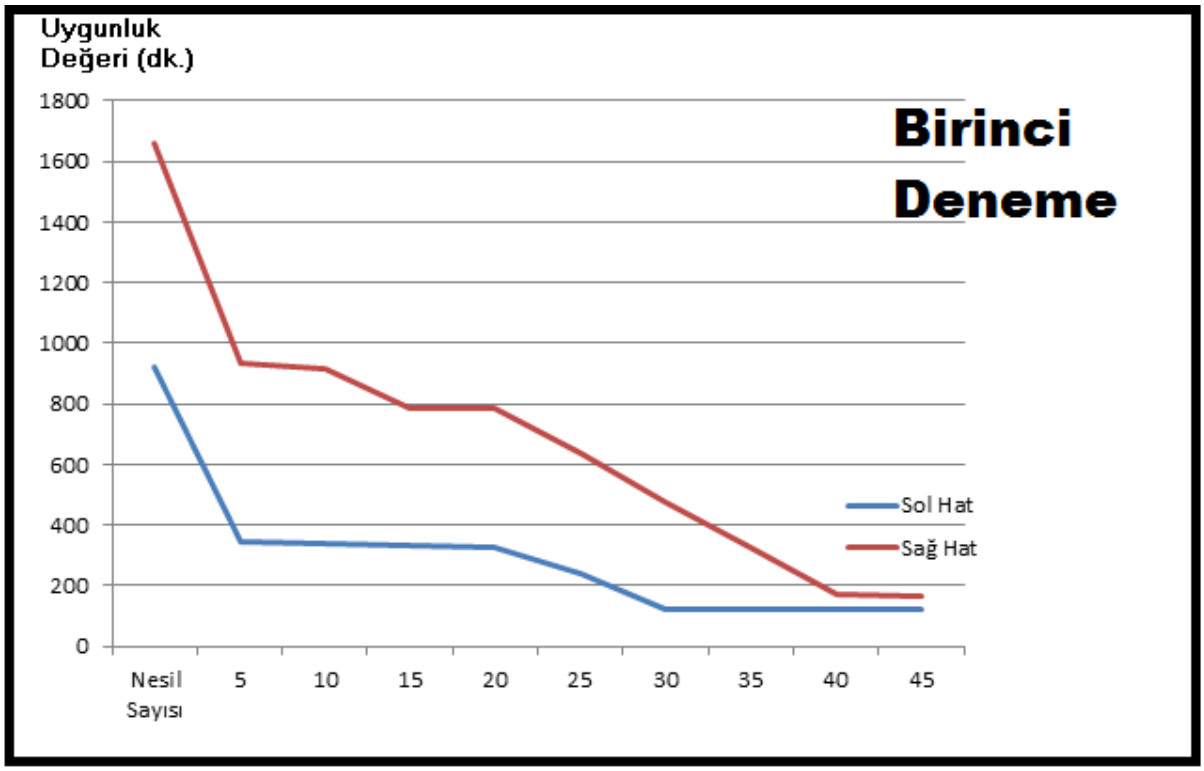
3.4.1. Birinci Deneme

- Elde edilen ekran görüntüsü



Şekil 3.4. Uygulama sonuçlarından birinci denemenin ekranı

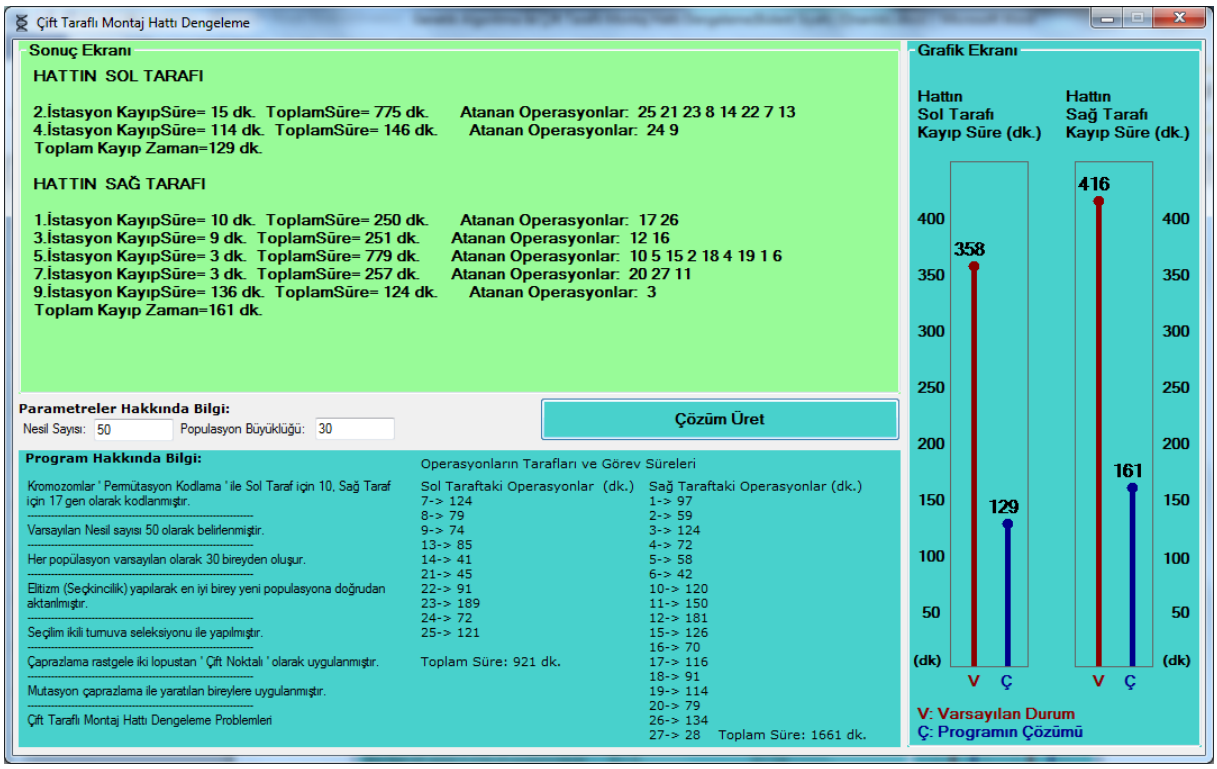
- Nesiller arası elde edilen uygunluk değerleri (Birinci Deneme)



Şekil 3.5. Uygulama sonuçlarından birinci denemenin uygunluk değerleri

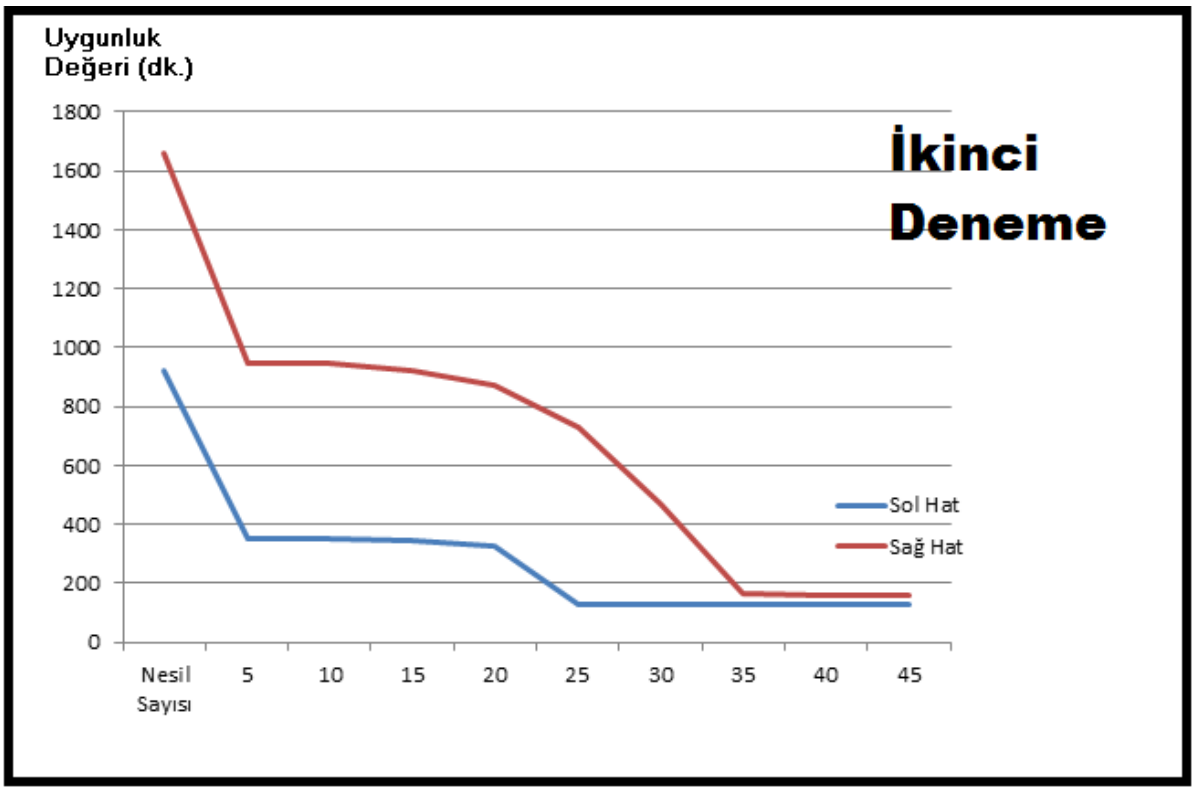
3.4.2. İkinci Deneme

- Elde edilen ekran görüntüsü



Şekil 3.6. Uygulama sonuçlarından ikinci denemenin ekranı

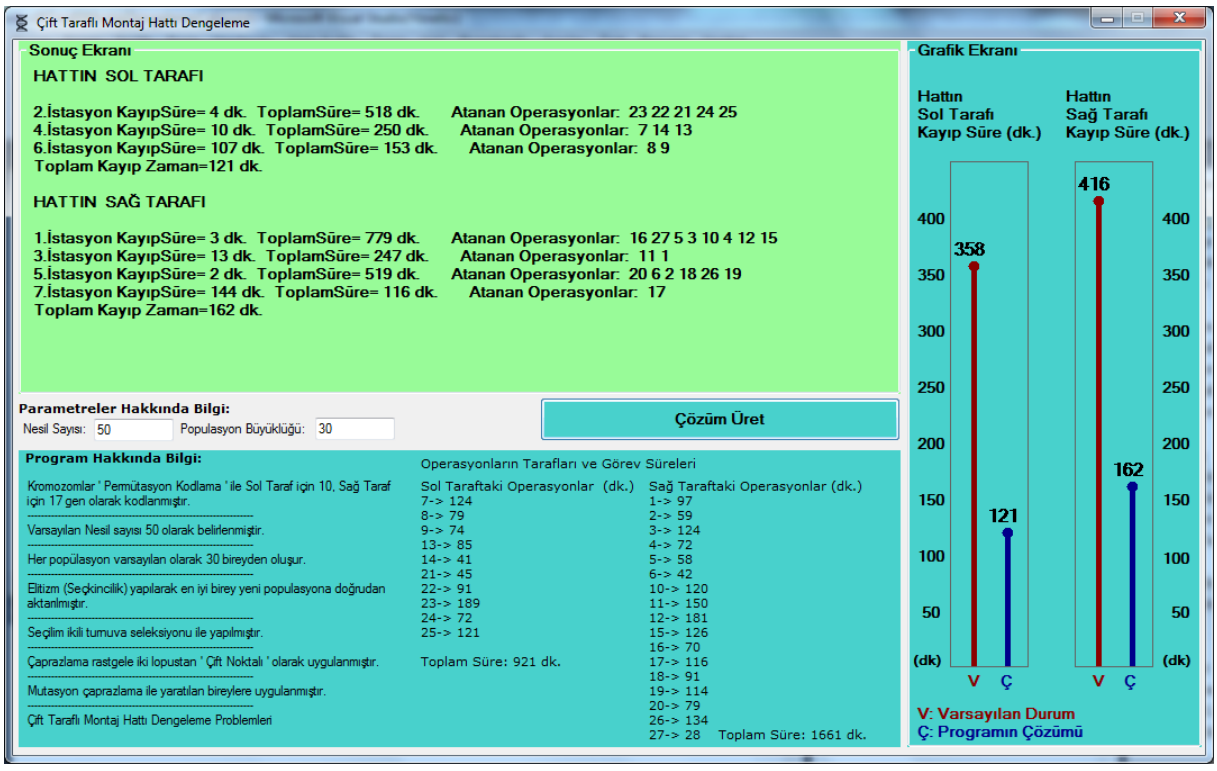
- Nesiller arası elde edilen uygunluk değerleri (İkinci Deneme)



Şekil 3.7. Uygulama sonuçlarından ikinci denemenin uygunluk değerleri

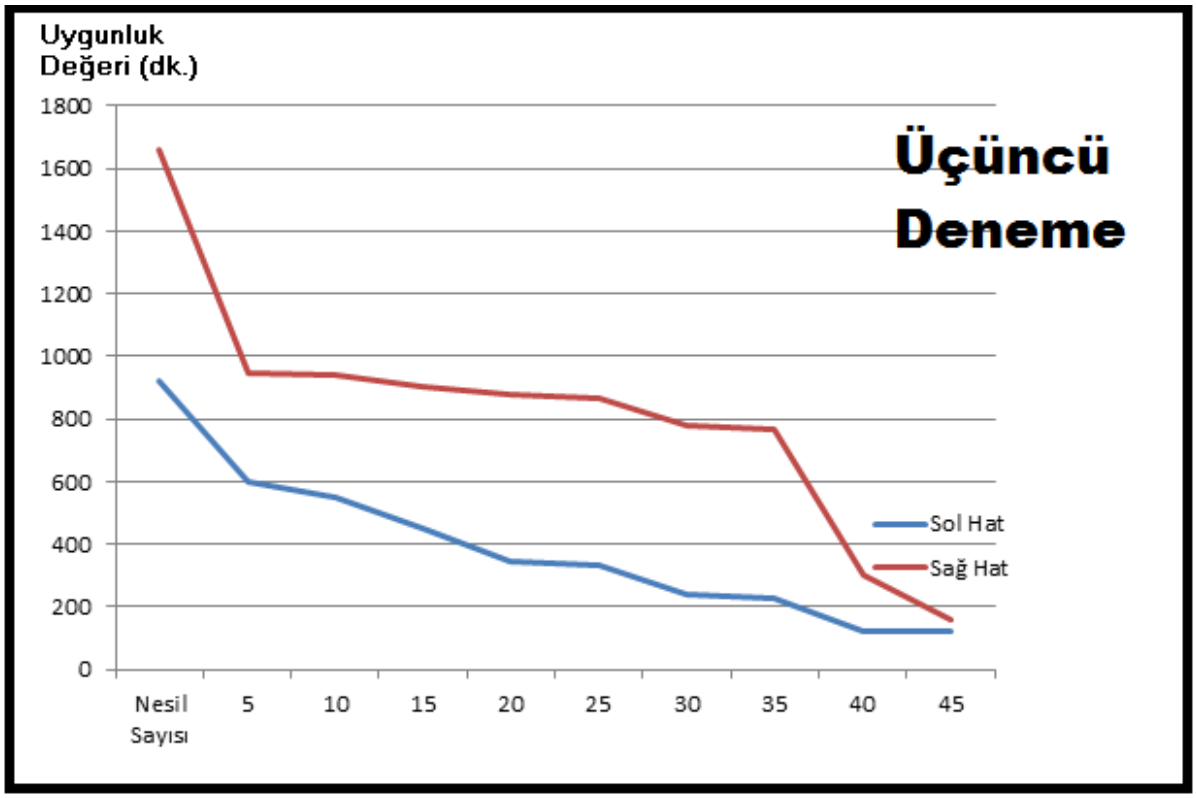
3.4.3. Üçüncü Deneme

- Elde edilen ekran görüntüsü



Şekil 3.8. Uygulama sonuçlarından üçüncü denemenin ekranı

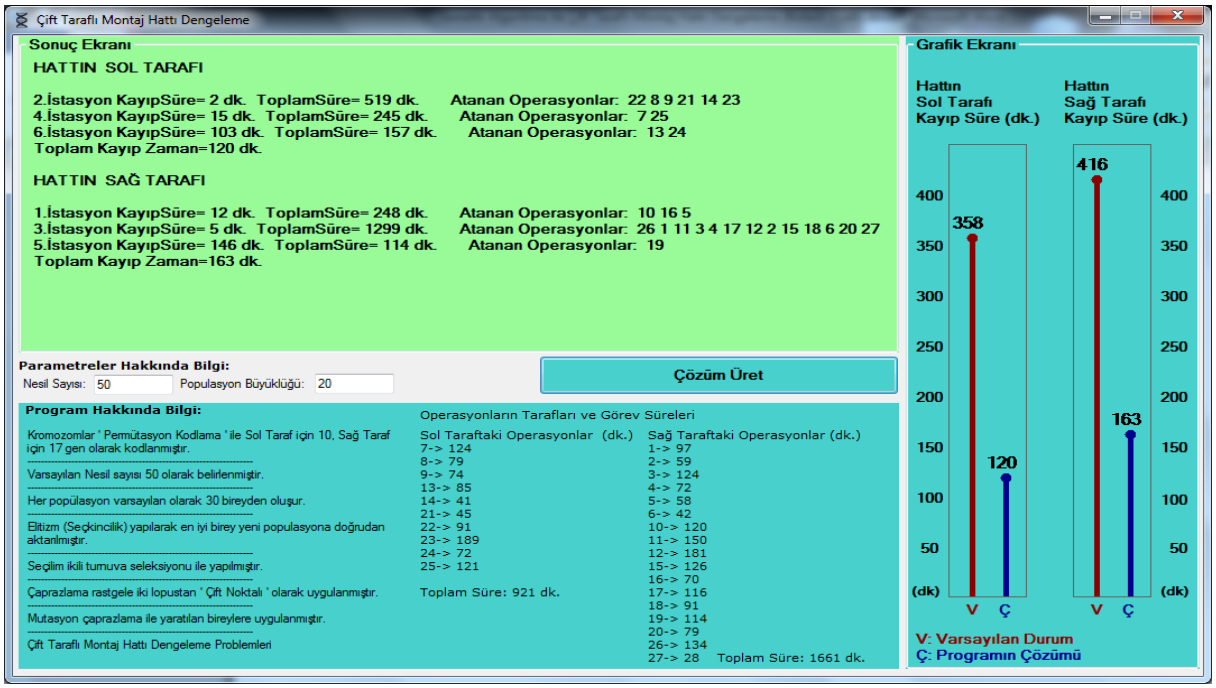
- Nesiller arası elde edilen uygunluk değerleri (Üçüncü Deneme)



Şekil 3.9. Uygulama sonuçlarından üçüncü denemenin uygunluk değerleri

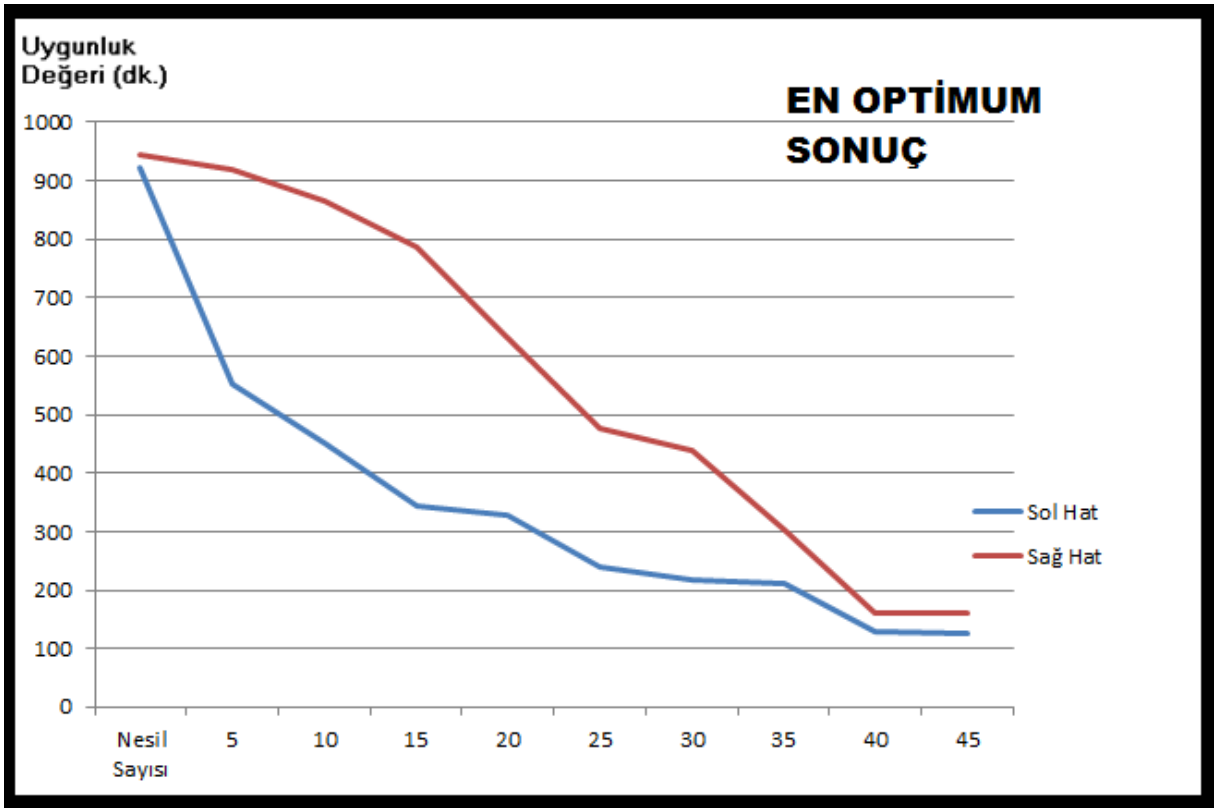
3.4.4. En Optimum Sonuç

- Elde edilen ekran görüntüsü



Şekil 3.9. Uygulama sonuçlarından en optimum sonucun bulunduğu ekran

- Nesiller arası elde edilen uygunluk değerleri (En Optimum Sonuç)



Şekil 3.10. Uygulama sonuçlarından en optimum sonucun uygunluk değerleri

4. UYGULAMANIN TÜM KODLARI

- Uygulamanın tüm kodları c# programlama dili ile kodlanmıştır :

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Collections;

namespace ciftTarafliMontajDengeleme
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        static int NesilSayisi;
        int BireySayisi;
        int GenSayisi;
        int SabitGenSayisi;
        int[] GenlerinSureleri;
        int[] ilkToplumBireyUygunlugu;
```

```
        ArrayList ilkToplumBireyUygunluguHesaplama = new ArrayList();
        Random Sayi = new Random();
        int[,] ilkToplum;
        int[,] YeniToplum;
        int OrtalamaCalismaSuresi;
        int ElitSayisi;
        int SolEsikDegeri;
        int SagEsikDegeri;
        String LblSolYazi;
        String LblSagYazi;
        int MutasyonSayisi;
        Bitmap bitmap;
        int istasyonlarToplamiSabitSol;
        int istasyonlarToplamiSabitSag;
        private void basla_Click(object sender, EventArgs e)
        {
            ilkDegerleriOlustur();
            for (int i = 1; i <= 2; i++)
            {
                if (i == 1)
                {
                    GenSayisi = 10;
                    GenlerinSureleri = new int[GenSayisi + 1];
                    ilkToplum = new int[BireySayisi + 1, GenSayisi + 1];
                    YeniToplum = new int[BireySayisi + 1, GenSayisi + 1];
                    GorevSureleriniGenlereYaz(i);
                    istasyonlarToplamiSabitSol = 358;
                    Genetik();
                }
                else
                {
                    GenSayisi = 17;
                    GenlerinSureleri = new int[GenSayisi + 1];
                    ilkToplum = new int[BireySayisi + 1, GenSayisi + 1];
                    YeniToplum = new int[BireySayisi + 1, GenSayisi + 1];
                    GorevSureleriniGenlereYaz(i);
                    istasyonlarToplamiSabitSag = 416;
                    Genetik();
                }
            }
        }
        public void ilkDegerleriOlustur()
        {
            try
            {
                NesilSayisi = Convert.ToInt32(txtNesil.Text);
                BireySayisi = Convert.ToInt32(txtBirey.Text);
            }
            catch
            {
                MessageBox.Show("Girilen Değerler Geçerli Değil.Rakam Giriniz.");
                Application.Exit();
            }

            if (NesilSayisi <= 1 || BireySayisi <= 1 || NesilSayisi >= 250 ||
BireySayisi >= 250)
            {
                MessageBox.Show("Girilen Değerler Geçerli Değil. Değerler '1' dan
Büyük ve 250 den Küçük Olmalıdır.");
                Application.Exit();
            }
            OrtalamaCalismaSuresi = 260;
```

```
            ElitSayisi = 2;
            ilkToplumBireyUygunlugu = new int[BireySayisi + 1];
            MutasyonSayisi = 1;
            SolEsikDegeri = 15;
            SagEsikDegeri = 18;
            LblSolYazi = "";
            LblSagYazi = "";
            SabitGenSayisi = 10;
            GrafikSol.Image = new Bitmap(GrafikSol.Width, GrafikSol.Height);
            GrafikSag.Image = new Bitmap(GrafikSag.Width, GrafikSag.Height);
        }
        public void GorevSureleriniGenlereYaz(int SolveyaSag)
        {
            int[] SolGorevSureleri = { 0, 124, 79, 74, 85, 41, 45, 91, 189, 72, 121 };
            int[] SagGorevSureleri = { 0, 97, 59, 124, 72, 58, 42, 120, 150, 181, 126,
70, 116, 91, 114, 79, 134, 28 };
            if (SolveyaSag == 1)
            {
                for (int j = 1; j <= SolGorevSureleri.Length - 1; j++)
                {
                    GenlerinSureleri[j] = SolGorevSureleri[j];
                }
            }
            else
            {
                for (int j = 1; j <= SagGorevSureleri.Length - 1; j++)
                {
                    GenlerinSureleri[j] = SagGorevSureleri[j];
                }
            }
        }
        public void Genetik()
        {
            ilkToplumOlustur();
            for (int ii = 1; ii <= NesilSayisi; ii++)
            {
                Uygunluk();
                Elitizm();
                Caprazlama();
                Mutasyon();
                ToplumDegistir();
                if (ii == NesilSayisi)
                {
                    SonUygunluk();
                    EkranCiz();
                }
            }
        }
        public void ilkToplumOlustur()
        {
            ArrayList randomSayilar = new ArrayList();
            for (int iii = 1; iii <= BireySayisi; iii++)
            {
                randomSayilar.Clear();
                for (int jjj = 1; jjj <= GenSayisi; jjj++)
                {
                    int randomSayi = Sayi.Next(1, GenSayisi + 1);
                    if (randomSayilar.IndexOf(randomSayi) == -1)
                    {
                        ilkToplum[iii, jjj] = randomSayi;
                        randomSayilar.Add(randomSayi);
                    }
```

```
                    else
                    {
                        jjj = jjj - 1;
                    }
                }
            }
        }
        public void Uygunluk()
        {
            ArrayList UygunlukistasyonDizileri = new ArrayList();
            int deger = 0;
            int esik;
            if (GenSayisi > SabitGenSayisi)
            {
                esik = SagEsikDegeri;
            }
            else
            {
                esik = SolEsikDegeri;
            }
            for (int k = 1; k <= BireySayisi; k++)
            {
                UygunlukistasyonDizileri.Clear();
                ilkToplumBireyUygunlugu[k] = 0;
                for (int l = 1; l <= GenSayisi; l++)
                {
                    UygunlukistasyonDizileri.Add(ilkToplum[k, l]);
                    deger = UygunlukDegerleri(UygunlukistasyonDizileri);
                    if (deger <= esik || l == GenSayisi)
                    {
                        ilkToplumBireyUygunlugu[k] = ilkToplumBireyUygunlugu[k] +
deger;
                        UygunlukistasyonDizileri.Clear();
                    }
                }
            }
        }
        public int UygunlukDegerleri(ArrayList GelenDiziKumesi)
        {
            int degertopla = 0;
            int degertoplakalan = 0;
            int degertoplabolum = 0;
            int uygunlukDegeri = 0;
            for (int kkk = 0; kkk < GelenDiziKumesi.Count; kkk++)
            {
                degertopla = degertopla +
GenlerinSureleri[Convert.ToInt32(GelenDiziKumesi[kkk])];
            }
            degertoplakalan = degertopla % OrtalamaCalismaSuresi;
            degertoplabolum = degertopla / OrtalamaCalismaSuresi;
            uygunlukDegeri = OrtalamaCalismaSuresi - degertoplakalan;
            uygunlukDegeri = uygunlukDegeri * (degertoplabolum + 1);
            if (GelenDiziKumesi.Count == GenSayisi)
            {
                uygunlukDegeri = degertopla;
            }
            return uygunlukDegeri;
        }
        public void Elitizm()
        {
            int EnKucuk, EnKucukindex;
            for (int elit = 1; elit <= BireySayisi; elit++)
```

```
            {
                EnKucuk = 100000;
                EnKucukindex = 0;
                int gecici = 0, gecici2 = 0;
                for (int elit2 = elit; elit2 <= BireySayisi; elit2++)
                {
                    if (ilkToplumBireyUygunlugu[elit2] < (EnKucuk))
                    {
                        EnKucuk = ilkToplumBireyUygunlugu[elit2];
                        EnKucukindex = elit2;
                    }
                }
                gecici = ilkToplumBireyUygunlugu[elit];
                ilkToplumBireyUygunlugu[elit] = ilkToplumBireyUygunlugu[EnKucukindex];
                ilkToplumBireyUygunlugu[EnKucukindex] = gecici;
                for (int degisenGenler = 1; degisenGenler <= GenSayisi;
degisenGenler++)
                {
                    gecici2 = ilkToplum[elit, degisenGenler];
                    ilkToplum[elit, degisenGenler] = ilkToplum[EnKucukindex,
degisenGenler];
                    ilkToplum[EnKucukindex, degisenGenler] = gecici2;
                }
            }
            for (int index = 1; index <= ElitSayisi; index++)
            {
                for (int bb = 1; bb <= GenSayisi; bb++)
                {
                    YeniToplum[index, bb] = ilkToplum[index, bb];
                }
            }
        }
        public void Caprazlama()
        {
            int galip1, galip2, rastgeleBirey1, rastgeleBirey2, rastgeleBirey3,
rastgeleBirey4, lopus1, lopus2, gecicilopus;
            for (int carpazi = ElitSayisi + 1; carpazi <= BireySayisi; carpazi =
carpazi + 2)
            {
                if (carpazi == BireySayisi)
                {
                    carpazi--;
                }
                rastgeleBirey1 = Sayi.Next(1, BireySayisi);
                rastgeleBirey2 = Sayi.Next(1, BireySayisi);
                rastgeleBirey3 = Sayi.Next(1, BireySayisi);
                rastgeleBirey4 = Sayi.Next(1, BireySayisi);
                lopus1 = Sayi.Next(2, GenSayisi - 1);
                lopus2 = Sayi.Next(2, GenSayisi - 1);
                if (lopus1 > lopus2)
                {
                    gecicilopus = lopus1;
                    lopus1 = lopus2;
                    lopus2 = gecicilopus;
                }
                if (rastgeleBirey1 > rastgeleBirey2)
                {
                    galip1 = rastgeleBirey2;
                }
                else
                {
                    galip1 = rastgeleBirey1;
```

```
                }
                if (rastgeleBirey3 > rastgeleBirey4)
                {
                    galip2 = rastgeleBirey4;
                }
                else
                {
                    galip2 = rastgeleBirey3;
                }
                for (int b = 1; b < lopus1; b++)
                {
                    YeniToplum[carpazi, b] = ilkToplum[galip1, b];
                    YeniToplum[carpazi + 1, b] = ilkToplum[galip2, b];
                }
                for (int b = (lopus1); b < lopus2; b++)
                {
                    YeniToplum[carpazi, b] = ilkToplum[galip2, b];
                    YeniToplum[carpazi + 1, b] = ilkToplum[galip1, b];
                }
                for (int b = (lopus2); b <= GenSayisi; b++)
                {
                    YeniToplum[carpazi, b] = ilkToplum[galip1, b];
                    YeniToplum[carpazi + 1, b] = ilkToplum[galip2, b];
                }
                for (int hataliBirey = carpazi; hataliBirey <= carpazi + 1;
hataliBirey++)
                {
                    ArrayList HataliGenDuzelt = new ArrayList();
                    for (int hataliGen = 1; hataliGen <= GenSayisi; hataliGen++)
                    {
                        if (HataliGenDuzelt.IndexOf(YeniToplum[hataliBirey,
hataliGen]) == -1)
                        {
                            HataliGenDuzelt.Add(YeniToplum[hataliBirey, hataliGen]);
                        }
                        else
                        {
                            HataliGenDuzelt.Add("*");
                        }
                    }
                    while (HataliGenDuzelt.Contains("*"))
                    {
                        for (int EksikGen = 1; EksikGen <= GenSayisi; EksikGen++)
                        {
                            int hataGen = HataliGenDuzelt.IndexOf("*");
                            if (HataliGenDuzelt.IndexOf(EksikGen) == -1)
                            {
                                HataliGenDuzelt.RemoveAt(hataGen);
                                HataliGenDuzelt.Insert(hataGen, EksikGen);
                            }
                        }
                    }
                    for (int arrayiGeneAta = 1; arrayiGeneAta <= GenSayisi;
arrayiGeneAta++)
                    {
                        YeniToplum[hataliBirey, arrayiGeneAta] =
Convert.ToInt32(HataliGenDuzelt[arrayiGeneAta - 1]);
                    }
                }
            }
        }
        public void Mutasyon()
```

```
        {
            int MutandBirey, mutgen1, mutgen2;
            String sabitgen1, sabitgen2;
            for (int mutand = 1; mutand <= MutasyonSayisi; mutand++)
            {
                MutandBirey = Sayi.Next(1, BireySayisi + 1);
                mutgen1 = Sayi.Next(0, GenSayisi);
                mutgen2 = Sayi.Next(0, GenSayisi);
                ArrayList MutasyonDuzelt = new ArrayList();
                for (int MutasGen = 1; MutasGen <= GenSayisi; MutasGen++)
                {
                    MutasyonDuzelt.Add(YeniToplum[MutandBirey, MutasGen]);
                }
                sabitgen1 = MutasyonDuzelt[mutgen1].ToString();
                sabitgen2 = MutasyonDuzelt[mutgen2].ToString();
                MutasyonDuzelt.RemoveAt(mutgen1);
                MutasyonDuzelt.Insert(mutgen1, sabitgen2);
                MutasyonDuzelt.RemoveAt(mutgen2);
                MutasyonDuzelt.Insert(mutgen2, sabitgen1);
                if (mutgen1 == mutgen2)
                {
                    mutand--;
                }
                for (int MutanGenAta = 1; MutanGenAta <= GenSayisi; MutanGenAta++)
                {
                    YeniToplum[MutandBirey, MutanGenAta] =
Convert.ToInt32(MutasyonDuzelt[MutanGenAta - 1]);
                }
            }
        }
        public void ToplumDegistir()
        {
            for (int toplumDegistirBirey = 1; toplumDegistirBirey <= BireySayisi;
toplumDegistirBirey++)
            {
                for (int toplumDegistirGen = 1; toplumDegistirGen <= GenSayisi;
toplumDegistirGen++)
                {
                    ilkToplum[toplumDegistirBirey, toplumDegistirGen] =
YeniToplum[toplumDegistirBirey, toplumDegistirGen];
                }
            }
        }
        public void SonUygunluk()
        {
            ArrayList UygunlukistasyonDizileri = new ArrayList();
            int deger = 0;
            int esik;
            int istasyonSayisi = 0;
            if (GenSayisi > SabitGenSayisi)
            {
                esik = SagEsikDegeri;
                istasyonSayisi = -1;
            }
            else
            {
                esik = SolEsikDegeri;
                istasyonSayisi = 0;
            }
            String gengrubu1 = "", gengrubu2 = "", gengrubu3 = "";
            int toplamGenDeger = 0;
            UygunlukistasyonDizileri.Clear();
```

```
            ilkToplumBireyUygunlugu[1] = 0;
            for (int l = 1; l <= GenSayisi; l++)
            {
                UygunlukistasyonDizileri.Add(ilkToplum[1, l]);
                deger = UygunlukDegerleri(UygunlukistasyonDizileri);
                if (deger <= esik || l == GenSayisi)
                {
                    ilkToplumBireyUygunlugu[1] = ilkToplumBireyUygunlugu[1] + deger;
                    istasyonSayisi = istasyonSayisi + 2;
                    toplamGenDeger = 0;
                    gengrubu2 = "";
                    for (int gecici_degisken = 0; gecici_degisken <
UygunlukistasyonDizileri.Count; gecici_degisken++)
                    {
                        toplamGenDeger = toplamGenDeger +
GenlerinSureleri[Convert.ToInt32(UygunlukistasyonDizileri[gecici_degisken])];
                        gengrubu2 = gengrubu2 + " " +
OperasyonlariDuzelt(UygunlukistasyonDizileri[gecici_degisken].ToString(), GenSayisi);

                    }
                    gengrubu1 = istasyonSayisi.ToString() + ".İstasyon KayıpSüre= " +
deger + " dk." + "  ToplamSüre= " + toplamGenDeger + " dk." + "       Atanan
Operasyonlar: ";
                    gengrubu3 = gengrubu3 + "\n" + gengrubu1 + gengrubu2;
                    UygunlukistasyonDizileri.Clear();
                }
                if (l == GenSayisi)
                {
                    ilkToplumBireyUygunluguHesaplama.Add(gengrubu3);
                }
            }
        }
        public void EkranCiz()
        {
            string ekranLabeli = "";
            ekranLabeli = ilkToplumBireyUygunluguHesaplama[0].ToString() + "\n" +
"Toplam Kayıp Zaman=" + ilkToplumBireyUygunlugu[1] + " dk.";
            if (GenSayisi > SabitGenSayisi)
            {
                LblSagYazi = "\n" + "HATTIN  SAĞ TARAFI" + "\n" + ekranLabeli;
                ResimCizdir(20, 450 - istasyonlarToplamiSabitSag, 1); //
                ResimCizdir(50, 450 - ilkToplumBireyUygunlugu[1], 2);//
            }
            else
            {
                LblSolYazi = "HATTIN  SOL TARAFI" + "\n" + ekranLabeli;
                ResimCizdir(20, 450 - istasyonlarToplamiSabitSol, 1);//
                ResimCizdir(50, 450 - ilkToplumBireyUygunlugu[1], 2); //
            }
            lblEkranYazi.Text = LblSolYazi + "\n" + LblSagYazi;
            ilkToplumBireyUygunluguHesaplama.Clear();
        }
        public void ResimCizdir(int x, int y, int Normal)
        {
            int eskix = x;
            int eskiy = 450;
            bitmap = (Bitmap)GrafikSol.Image;
            Graphics g = Graphics.FromImage(bitmap);
            if (GenSayisi > SabitGenSayisi)
            {
                bitmap = (Bitmap)GrafikSag.Image;
                g = Graphics.FromImage(bitmap);
```

```
            }
            if (Normal == 1)
            {
                g.DrawString((450 - y).ToString(), new Font("Microsoft Sans Serif",
12), new SolidBrush(Color.Black), x - 20, y - 25);
                g.FillEllipse(new SolidBrush(Color.DarkRed), x - 5, y - 5, 10, 10);
                g.DrawLine(new Pen(Color.DarkRed, 5), (float)eskix, (float)eskiy,
(float)x, (float)y);
            }
            else
            {
                g.DrawString((450 - y).ToString(), new Font("Microsoft Sans Serif",
12), new SolidBrush(Color.Black), x - 20, y - 25);
                g.FillEllipse(new SolidBrush(Color.DarkBlue), x - 5, y - 5, 10, 10);
                g.DrawLine(new Pen(Color.DarkBlue, 5), (float)eskix, (float)eskiy,
(float)x, (float)y);
            }
            if (GenSayisi > SabitGenSayisi)
            {
                GrafikSag.Image = bitmap;
            }
            else
            {
                GrafikSol.Image = bitmap;
            }
        }
        public String OperasyonlariDuzelt(String OperasyonNumarasi, int GenSayisi)
        {
            String[] SolOperatorler = { "0",
"7","8","9","13","14","21","22","23","24","25"};
            String[] SagOperatorler = { "0",
"1","2","3","4","5","6","10","11","12","15","16","17","18","19","20","26","27" };
            String DonusDegeri = "";
            if (GenSayisi > SabitGenSayisi)
            {
                DonusDegeri=SagOperatorler[Convert.ToInt32(OperasyonNumarasi)];
            }
            else
            {
                DonusDegeri = SolOperatorler[Convert.ToInt32(OperasyonNumarasi)];
            }
            return DonusDegeri;
        }
    }
}
```