İÇİNDEKİLER

1.Giriş

1930 yılların başında ilk olarak Karl Menger tarafından matematiksel bir ifade olarak Gezgin Satıcı Problemi (GSP) tanımlanmıştır. Literatür içerisinde gezgin satıcı problemi NP-zor (NPhard) olarak yer almaktadır. Günümüzde bilimsel yaklaşımlar sonucu elde edilen veriler sayesinde işletmeler, maliyet ve rekabet konularında üstünlük ve avantaj sağlayabilmektedirler. Genetik Algoritmalar (GA) Holland tarafından ilk kez 1975 yılında önerilmiş ve biyolojik sistemler içerisindeki evrim sürecine benzeyen arama ve optimizasyon yöntemi olarak çıkmıştır[1]. GSP üzerinde çok fazla çalışma yapılan kombinasyonal eniyileme problemlerinin başında gelmektedir. Gezgin satıcı probleminin genetik algoritmalar ile en iyi çözümü bulma isteği de buradan kaynaklanmıştır. Problemlerin çözümünde kombinasyonal optimizasyon kullanılarak çözüme ulaşmada sıklıkla kullanılan yöntemlerden biri olarak karşımıza genetik algoritmalar çıkmaktadır.

Bu çalışmada meta-sezgisel bir yöntem olan genetik algoritmalardan faydalanılarak gezgin satıcı problemine çözüm aranmıştır. Yapılan literatür taramaları sonucu örnek olarak lelana.com adresinden (Traveling Salesman Problem) [2] TSP projesi gezgin satıcı problemi ve kodlarının bir bölümü alınarak kendimize ait gezgin satıcı problemi uygulaması yapılmıştır. Çalışma sonucunda projenin kaynak kodları paylaşıma açılarak problemin çözümünde farklı yolların geliştirilmesinin sağlanacağı düşünülmektedir.

Çalışmanın ikinci bölümünde gezgin satıcı probleminin matematiksel ifadelerine yer verilerek problem ile ilgili çözümler ve literatür özeti paylaşılacaktır. Üçüncü bölümde problemin çözümünde kullanılan genetik algoritma ele alınacak ve adımlar incelenecektir. Dördüncü bölümde tasarımı ve kodlaması yapılan GSP uygulaması hakkında detaylı açıklamalara yer verilecektir. Son olarak elde edilen sonuçlar özetlenerek daha sonraki yapılacak çalışmalar için önerilerde bulunulacaktır.

2.Gezgin Satıcı Problemi

Gezgin satıcı probleminde bir satış elemanının veya bir aracın belirli bir noktadan başlayarak sistem içerisinde yer alan diğer tüm şehirlere yalnızca bir kez uğramak şartı ile başladığı noktaya geri dönmesi amaçlanmış olup, geri dönme sırasında yapmış olduğu toplam tur sayısını veya maliyetini minimuma indirgenmesi amaçlanmaktadır. Toplam nokta sayısını n adet olarak ele alırsak birinci nokta için (n-1) adet, ikinci nokta için (n-2) adet, üçüncü nokta için (n-3) adet, dördüncü nokta için(n-4) adet ve n' e kadar bu şekilde devam edecek nokta vardır. Burada n problemin boyutunu ifade etmektedir. Problem için olurlu tur sayısı (n-1)! Adet olarak çıkmaktadır. Buradan hareketle nokta sayısı az olduğu durumlarda kesin çözüme ulaşmak kolay ve mümkün iken, nokta sayısı, problemin boyutu büyüdükçe alternatif çözüm sayısı da hızla artmakta ve optimum çözüm bulunabilmesi için çok fazla zaman harcanması gerekmektedir. Bu nedenle sezgisel ve meta sezgisel yöntemler kullanılarak kısa sürede en iyi veya en iyi ye yakın sonuçlar elde edilebilmektedir. Gezgin satıcı probleminin matematiksel olarak ifadesi aşağıdaki gibidir:

Minimize: $z= \sum_{i=1}^{n} \sum_{j=1, i\neq j}^{n} x(i,j)d(i,j)$ (1)

Kısıtlar:

$$\sum_{j=1, j\neq i}^{n} x(i,j) = 1, i = 1,2,\dots,n \quad (2)$$

$$\sum_{i=1, i\neq j}^{n} x(i,j) = 1, j = 1,2,\dots,n \quad (3)$$

$$\sum_{i,j\in S, i\neq j} x(i,j) \leq |S| - 1, \forall S \quad \{1,2,\dots,n\} \quad (4)$$

$$x(i, j) = \begin{cases} 1, i\ noktasından\ j\ noktasına\ gidiliyor\ ise \\ 0, i\ noktasından\ j\ noktasına\ gidilmiyor\ ise \end{cases} \quad (5)$$

GSP' nin amaç fonksiyonu 1 numaralı eşitlikte verilmektedir. Burada d(i, j) ifadesi i ve j noktaları arasında olan mesafeyi göstermektedir. X(i, j) ise i noktasından j noktasına gidilip gidilmediğini ifade etmektedir. 2 ve 3 numaralı eşitlikler ise her bir noktaya yani şehir e yalnızca bir defa uğranılacağını garanti altına almaya yöneliktir. Açacak olursa eğer 2 numaralı eşitlikte her noktadan yalnızca bir defa çıkılacak, 3 numaralı eşitlikte de her noktaya yalnızca bir kez gidilecektir. 4 numaralı eşitlik ise oluşabilecek alt turlardan kurtulmaya yönelik olan alt tur eleme formülsayonudur. 5 numaralı eşitlikte x(i, j)' nin 1 olması i noktasından j noktasına gidildiğini, 0 olması ile de i noktasından j noktasına gidilmediğini göstermektedir.

Literatürde GSP için kesin çözüm algoritmalarının bazıları şu şekildedir; dal ve sınır yöntemi, dinamik programlama ve dal kesme algoritmaları şeklindedir. Bu algoritmalar belirli bir boyuta kadar sorunsuz çalışıyor olsalar da GSP' nin nokta sayısı arttıkça daha doğrusu problemin boyutu veya şehir sayısı arttıkça buna bağlı olarak bilgisayarın işlem

zamanı artacağından optimum sonuca ulaşmak imkansız bir hal almaktadır. Bu nedenle sezgisel ve meta sezgisel yöntemler kullanmaya yönelerek, K-opt, V-opt, yasaklı arama, tavlama benzetimi, genetik algoritmalar, karınca kolonileri, yapay sinir ağları gibi yöntemler geliştirilmiştir. Bu yöntemler optimum sonucu vermekle beraber problemin kısa çözüm süresi ile de kullanıcılara yardımcı olmaktadır.

3.Genetik Algoritmalar ve Gezgin Satıcı Problemi

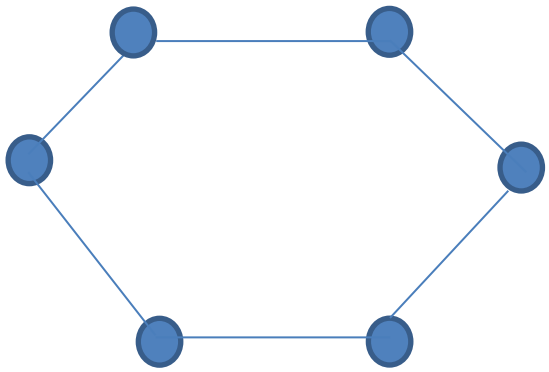
3.1.Genetik Algoritmalar

Genetik algoritmalar klasik yöntemlerle çözümü zor olan problemlerin çözümünü bulmak amacı ile biyolojik evrim süreçlerinin bilgisayar ortamında simülasyonunun sağlandığı ve kodlandığı, bu evrim süreçlerini kullanarak en iyi veya en iyiye yakın değerlerin araştırıldığı bir arama yöntemidir. Genetik algoritmalarda öncelikle istenilen problemin parametrelerine uygun şekilde kodlaması yapılarak kromozomlar oluşturulur. Devamında oluşturulan kromozomlardan meydana gelen bir başlangıç popülasyonu oluşturulur. Popülasyonda bulunan her bir kromozom problemin çözümü için aday konumundadır. Bu işlemlerin ardından doğadaki genetik süreçlere dayanan "çaprazlama" ve "mutasyon" işlemleri uygulanarak daha iyi kromozomlar elde edilmeye çalışılır. Arka arkaya gerçekleştirilen iterasyon ile yeni nesiller oluşturulur ve bu yeni nesillerin verdiği uygunluk değerleri popülasyondakiler ile karşılaştırılır. Elde edilen yeni nesiller içerisindeki kromozomlar popülasyon içerisindeki kromozomlar ile karşılaştırılır ve daha iyi olan kromozomlar popülasyona dahil edilir. Bu şekilde devam eden evrim sürecinde popülasyondaki kromozomlar yerlerini bir sonraki daha iyi kromozomlara bırakırlar. Bu işlem nesil sayısının istenilen sayıya ulaşıncaya kadar veya popülasyondaki uygunluk değerlerinde iyileşmenin en aza indiği zamana kadar devam eder.

GSP var olan her şehre uğrayarak ve tekrar başlangıç şehrine geri dönmek şartı ile diğer şehirleri en kısa yolu izleyerek dolaşma prensibine dayandığından bahsetmiştik. Örneğin 5 şehirli bir kümemiz olsun. Yol sayısı (n-1)! İle hesaplanmaktadır ve bu 5 şehir için var olan 24 olası yol sonucu çıkmaktadır. Bu örnekte 24 olası yolun her birinin uzunluklarını hesaplayıp en kısa yolu bularak çözüme ulaşabilirsiniz fakat şehir sayısı arttığında bu çözüm bizim için içinden çıkılamaz karmaşık bir hale gelmektedir.

Örnek: 6 şehir ile yapılan gezgin satıcı örneği

(n-1)! Uygulandığında 120 şehir yolu hesaplanır.



3.2.Gezgin Satıcı Problemi İçin Kromozom Yapısı

Genetik algoritmalarda en fazla kullanılan kodlama yöntemi ikili kodlama yöntemi ve permütasyon kodlama yöntemidir. İkili kodlama yönteminde her kromozom 0 ve 1' lerden oluşan bir diziyi ifade ederken, permütasyon kodlama yönteminde her kromozom kendisini oluşturan karakterlerin sırasını göstermektedir. Gezgin satıcı probleminin kromozomlarının kodlanması permütasyon yöntemine göre yapılmaktadır.

Noktaları izlerken yani şehirleri izlerken izlemiş olduğumuz rotaların her birini birer kromozom olarak temsil edilmektedir. Buna göre, 6 şehir içeren bir popülasyon için:

(402513) – (105243) – (13542) gibi farklı gezinme sıralarını ifade eden kromozomlar olacaktır.

3.3.Başlangıç Popülasyonunun Oluşturulması

Genetik algoritmalarda ilk aşamada yeni nesillerin türetilebileceği bir başlangıç popülasyonunun oluşturulması gerekmektedir. Klasik genetik algoritmalarda başlangıç popülasyonu genel olarak rastgele oluşturulur.

Popülasyonda, en kısa mesafeyi sağlayan, çözüm için en uygun olan kromozomları seçerek ikişerli çaprazlama ile yeni bireyler elde edilir. Kaynak olarak alınan örnekte başlangıç parametreleri Tablo-1 de verilmiştir.

Tablo 1.Gezgin satıcı problemi başlangıç parametreleri

Parametreler	Değerler
Popülasyon Büyüklüğü	10,000
Grup Büyüklüğü	5
Mutasyon	3%
Yakın Şehirler	5
Yakın Şehir Olasılığı	90%

3.4.Gezgin Satıcı Problemi İçin Uygunluk Fonksiyonu

Başlangıç popülasyonun oluşturulmasından sonraki aşama bu popülasyonda yer alan kromozomların uygunluk değerlerinin hesaplanması olacaktır. Gezgin satıcı problemi için uygunluk değeri, ziyaret edilecek olan şehirlerin kromozomdaki sıraya uygun olarak mesafelerinin toplanması ile elde edilir. Çaprazlama işleminin ardından yeni kromozomlar elde edildiğinde her bir kromozomun uygunluk değeri de benzer şekilde hesaplanabilmektedir.

3.5.Seçim Süreci

Yeni kromozomların oluşturulabilmesi için başlangıç popülasyonu içerisinden iki ebeveyn seçilir ve bu ebeveynler çaprazlanarak yeni bireyler oluşturulur. Burada soru olarak karşımıza şu çıkmaktadır. Popülasyon içerisinden hangi bireyler ebeveyn olarak seçilmelidir?

Bunun için çeşitli yöntemler mevcuttur. Bu yöntemlerden en çok kullanılanları rulet tekerleği, sıralama ve turnuva yöntemleridir.

3.6.Çaprazlama

Çaprazlama işlemi genetik algoritmalar için en önemli parametrelerden bir tanesidir. Seçim işleminin ardından bireyler çaprazlama işlemine tabi tutularak yeni bireyler elde edilir. Buradaki amaç ebeveyn kromozom genlerinden uygunluk değeri daha yüksek genler, kromozomlar meydana getirmektir. Tek nokta ve çift nokta çaprazlama yöntemleri bulunmaktadır. Bu iki yöntem dışında da çaprazlama teknikleri vardır. Bu yöntemde ebeveynler için ortak bir çaprazlama noktası belirlenerek çaprazlama işlemi gerçekleştirilir.

3.7.Mutasyon

Genetik algoritmalarda mutasyon popülasyonda genetik çeşitliliği sağlamak ve korumak için kullanılan bir operatördür. Mutasyon işlemi sonucunda çocuk genlerinden bir ya da bir kaçı rastgele olarak değiştirilir ve yeni kromozomlar meydana gelir. Gezgin satıcı probleminde mutasyon işlemi rastgele iki genin seçilerek yerlerinin değiştirilmesi şeklinde yapılır.

Örneğin;

Mutasyon Uğramamış Kromozom1: 53102

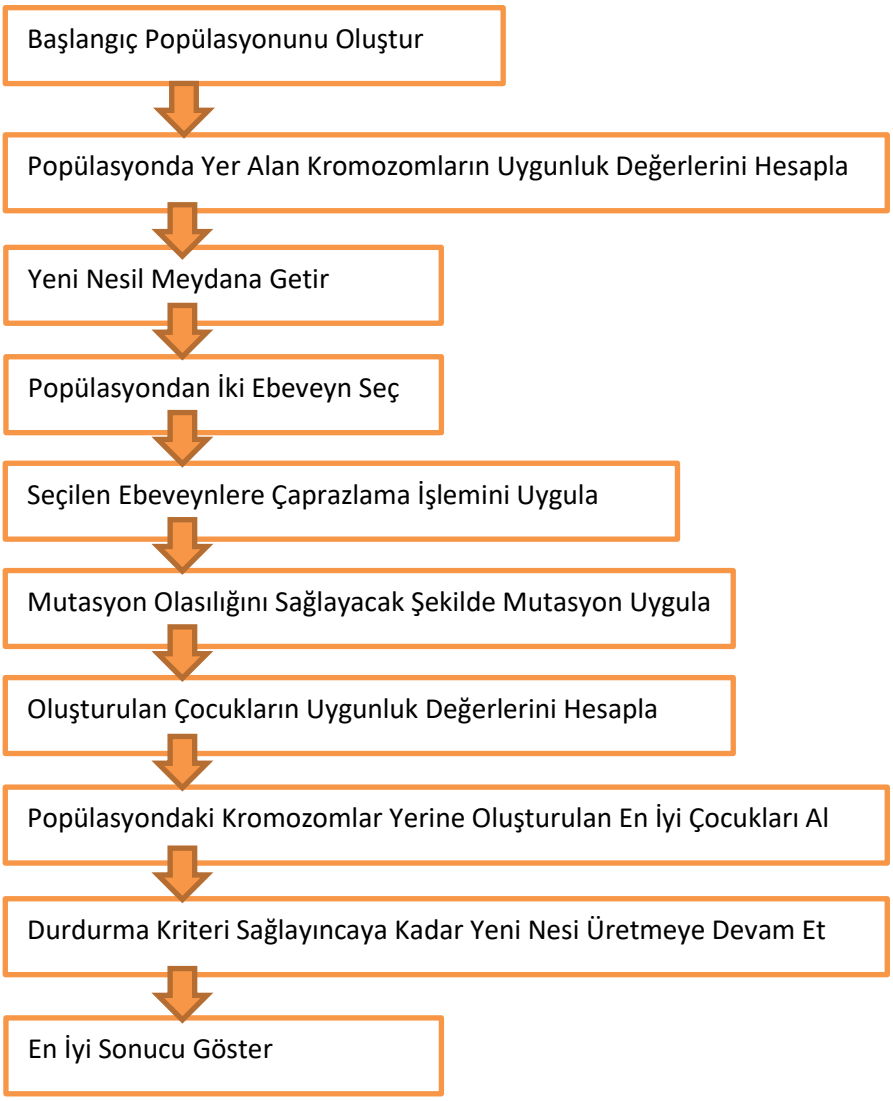
Mutasyon Uğramış Kromozom1: 03152

Bu örnekte kromozom1'in birinci ve dördüncü genleri rastgele seçilmiş iki gen olduğunu varsayarak kromozom2 de birinci ve dördüncü genleri yer değiştirerek kromozom2 mutasyona uğratılmış olunur.

3.8.Yeni Neslin Oluşturulması

Başlangıç popülasyonundan seçilen ebeveynler çaprazlama ve mutasyon operatörleri kullanılarak yeni aday sonuçları olan çocuklar elde edilir. Bu noktada çocukların uygunluk değerleri ile popülasyondaki kromozomların uygunluk değerleri karşılaştırılır. Eğer buradaki karşılaştırma sonucunda çocuk genlerin uygunluk değeri popülasyondaki kromozomlardan daha iyi ise popülasyondaki kromozomlar çıkarılır ve yerlerine yeni nesil çocuklar alınır. Bu işlem önceden belirlenen nesil sayısına ulaşıncaya kadar devam eder.

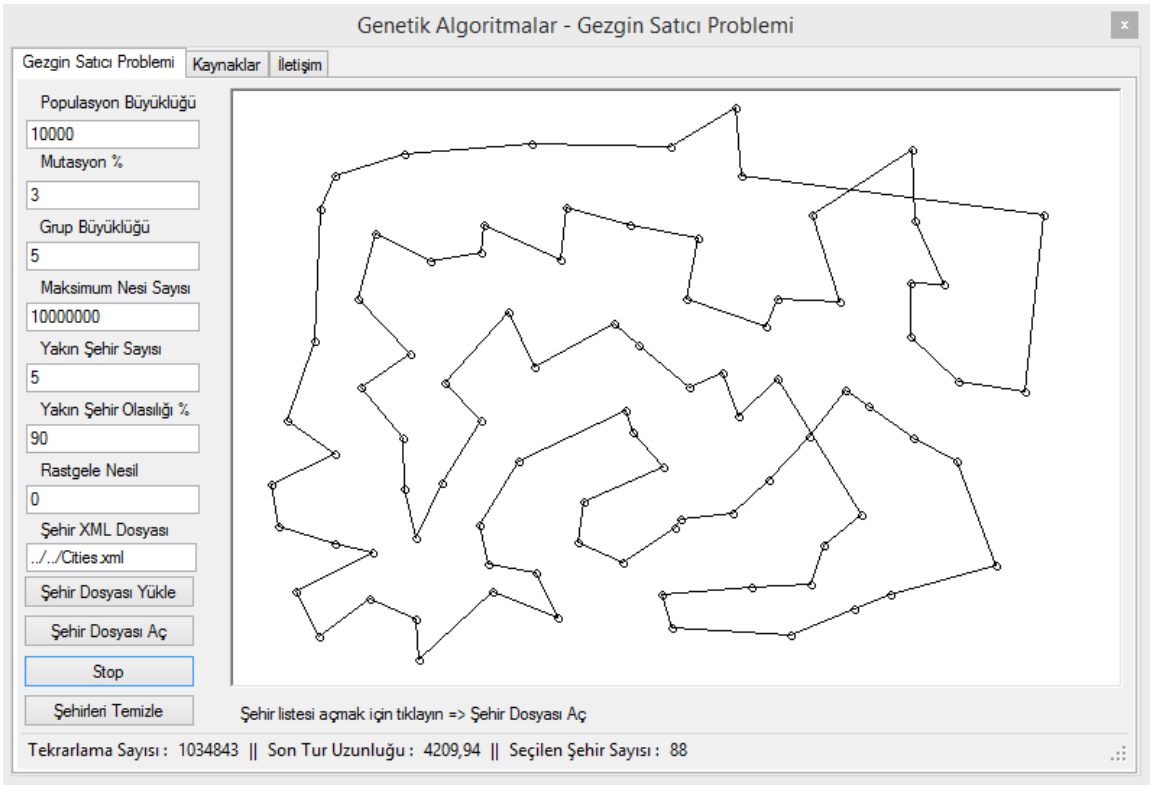
3.9.Genetik Algoritma Yapay Kodu



Yukarıdaki akış diyagramında genetik algoritma çalışma prensibi şematize edilmiştir.

4.Gezgin Satıcı Problemi Uygulamasının Açıklanması

GSP uygulamasının kaynak kodları lalena.com adresinden indirilmiştir. Program Visual Basic dilinde hazırlanmıştır ve 5 parametreye göre en iyi gezici rotasını hesaplamaktadır. Görsel olarak hazırlanan yazılımda kullanıcıdan şehirlerin belirlenmesi istenmekte ya da koordinatları XML olarak kayıtlı şehir listeleri program içerisine alınarak şehirlerin rotaları hesaplanabilmektedir. C# programlama diline çevrilen yazılım ile programın kullanıcı arayüzü tekrar düzenlenmiştir. Programın kaynak kodları içerisinde gerekli açıklamalar yapılarak anlatılmıştır. Şekil-1 de gezgin satıcı problemi uygulamasının kullanıcı arayüzü görülmektedir.



Şekil 1. Gezgin Satıcı Problemi Uygulaması Ekran Görüntüsü

```
using System;
using System.Collections.Generic;
using System.Text;

namespace GezginSaticiProblemi
{
    public class Link
    {
        // birinci nokta bağlantısı
        private int connection1;
        // birinci şehir bağlantısı
        public int Connection1
        { get
            { return connection1;
            }
            set
            { connection1 = value; ;
            }
        }
        //ikinci nokta bağlantısı
        private int connection2;
        //ikinci şehir bağlantısı
        public int Connection2
        { get
            { return connection2;
            }
            set
            { connection2 = value;
            }
        }
    }
}
```

Şekil 2. Şehirler arası bağlantı sınıfı

```
namespace GezginSaticiProblemi
{
    class Population : List<Tour> // populasyon class ı tour listesinden miras alır.
    {
        private Tour bestTour = null; // besttour değerine null değeri atanır.
        public Tour BestTour   // get ve set blokları ile değer okunması ve yazılması sağlanır.
        { set
            {  bestTour = value;
            }
            get
            {  return bestTour;
            }
        }
        public void CreateRandomPopulation(int populationSize, Cities cityList, Random rand, int chanceToUseCloseCity)
        {//turlar için rastgele başlangıç kümesi oluşturulur.
            int firstCity, lastCity, nextCity;
            for (int tourCount = 0; tourCount < populationSize; tourCount++)
            {  // tour class ını oluşturara şehir liste sayısını buna atanır.
                Tour tour = new Tour(cityList.Count);
                // tur için başlangıç noktası oluşturulur.
                firstCity = rand.Next(cityList.Count); //birinci şehire rastgele şehir listesi sayısı atanır.
                lastCity = firstCity; // önceki şehir birinci şehire atanır.
                for (int city = 0; city < cityList.Count - 1; city++)
                { do
                    { // bir sonraki şehir için rastgele şehir listesi tutulur.
                        if ((rand.Next(100) < chanceToUseCloseCity) && (cityList[city].CloseCities.Count > 0))
                        { // %75 olasılıkla birbirne yakın şehirleri bulur.
                            nextCity = cityList[city].CloseCities[rand.Next(cityList[city].CloseCities.Count)];
                        }
                        else
                        { // eğer yukarıdaki işlem gerçekleşmezse rastgele bir şehir seçilir.
                            nextCity = rand.Next(cityList.Count);
                        }
                        // şehirlerin turları ve bağlatıları kontrol edilir.
                    } while ((tour[nextCity].Connection2 != -1) || (nextCity == lastCity));
                    // a şehrinden b ye giderken [1] için A = B ve [1]  B = A olur.
                    tour[lastCity].Connection2 = nextCity; // önceki şehir bağlantısı sonraki şehir bağlantısına atanır.
                    tour[nextCity].Connection1 = lastCity; // sonraki şehir bağlantısı önceki şehir bağlantısına atanır.
                    lastCity = nextCity; // bir önceki şehir sonraki şehir olmuş olur.
                }
                // iki şehir birbirne bağlanır.
                tour[lastCity].Connection2 = firstCity;
                tour[firstCity].Connection1 = lastCity;
                tour.DetermineFitness(cityList);
                Add(tour); // tur a eklenir
                if ((bestTour == null) || (tour.Fitness < bestTour.Fitness))
                {  BestTour = tour; // tur değeri null sa veya  tur değeri bulunan en iyi tur değerinden küçükse
                } // en iyi tur değerine bulunan tur atanır.
            }
        }
    }
}
```

Şekil 3. Popülasyon sınıfı

Şehirler sınıfı ile şehirlerin oluşturulması konularının alınması ile ilgili işlemler yapılmıştır. Her şehir e ait iki bağlantı bulunmaktadır. Bu bağlantılar programlama sırasında da açıklanmıştır. Önceki ve sonraki bağları olarak rastgele dağılım içerisinde her noktadan bir çıkış ve yalnızca bir girişin olacağı şekilde programlaması yapılmıştır. Bu sayede gezgin satıcı problemine uygun çözüm sonucuna ulaşılmıştır. Şehirleri tekrar etmeden en iyi sonucu veren genetik algoritma ile problem çözülmüştür.

Bağlantı sınıfı ile şehirler arası ikili bağlantılar oluşturulmuştur bu sayede her şehirden giriş ve çıkış olmak üzere iki bağlantı yer almış bulunmaktadır.

```
namespace GezginSaticiProblemi
{
    public class City
    {  //şehirlerin x ve y konumları
        public City(int x, int y)
        {Location = new Point(x, y); //konum belirlenir.
        }
        private Point location; // point lokasyonu ile lehirlerin konum değerleri okunur ve yazılır.
        public Point Location
        {  get
            { return location;
            }
            set
            { location = value;
            }
        }
        private List<double> distances = new List<double>();
        public List<double> Distances // distances adında liste tanımlanır.
        { get
            { return distances;
            }
            set
            { distances = value;
            }
        }
        private List<int> closeCities = new List<int>();
        public List<int> CloseCities // closeCities adında liste tanımlanır.
        { get
            { return closeCities;
            }
        }
        public void FindClosestCities(int numberOfCloseCities) // bulunan şehirler metodu
        {   double shortestDistance;
            int shortestCity = 0;
            double[] dist = new double[Distances.Count];
            Distances.CopyTo(dist);
            if (numberOfCloseCities > Distances.Count - 1)
            { numberOfCloseCities = Distances.Count - 1;
            }
            closeCities.Clear();
            for (int i = 0; i < numberOfCloseCities; i++)
            { shortestDistance = Double.MaxValue;
                for (int cityNum = 0; cityNum < Distances.Count; cityNum++)
                { if (dist[cityNum] < shortestDistance)
                    { shortestDistance = dist[cityNum];
                        shortestCity = cityNum;
                    }
                }
                closeCities.Add(shortestCity);
                dist[shortestCity] = Double.MaxValue;
            }
```

Şekil 4.City Sınıfı

```
class Tsp
{public delegate void NewBestTourEventHandler(Object sender, TspEventArgs e);
    public event NewBestTourEventHandler foundNewBestTour;
    Random rand;// rastgelen değer üretecek rand nesnesi tanımlanır.
    Cities cityList;
    Population population;
    private bool halt = false;
    public bool Halt
    { get
        { return halt;
        }
        set
        { halt = value;
        }
    }
    public Tsp()
    {
    }
    //populationSize başlangıçtan önce rastgele tur sayısının büyüklüğü.
    //maxGenerations maksimum oluşan nesil sayısı.
    // groupSize nesillerin grup içerisindeki büyüklükleri.
    // mutation çocukların mutasyon olasılıkları.
    // seed rastgele sayı.
    // chanceToUseCloseCity yakın şehirler arası olasılık.
    // cityList şehir listelerindeki turlar.
    public void Begin(int populationSize, int maxGenerations, int groupSize, int mutation, int seed, int chanceToUseCloseCity, Cities cityList)
    {rand = new Random(seed);
        this.cityList = cityList;
        population = new Population();
        population.CreateRandomPopulation(populationSize, cityList, rand, chanceToUseCloseCity);
         displayTour(population.BestTour, 0, false);
         bool foundNewBestTour = false;
         int generation;
         for (generation = 0; generation < maxGenerations; generation++)
         { if (Halt)
             { break;
             }
             foundNewBestTour = makeChildren(groupSize, mutation);
             if (foundNewBestTour)
             { displayTour(population.BestTour, generation, false);
             }
         }
         displayTour(population.BestTour, generation, true);
    }
    bool makeChildren(int groupSize, int mutation)
    {   int[] tourGroup = new int[groupSize];
        int tourCount, i, topTour, childPosition, tempTour;
        for (tourCount = 0; tourCount < groupSize; tourCount++)
        { tourGroup[tourCount] = rand.Next(population.Count);
        }
        for (tourCount = 0; tourCount < groupSize - 1; tourCount++)
        { topTour = tourCount;
            for (i = topTour + 1; i < groupSize; i++)
            { if (population[tourGroup[i]].Fitness < population[tourGroup[topTour]].Fitness)
                {
                    topTour = i;
                }
            }
            if (topTour != tourCount)
            {
                tempTour = tourGroup[tourCount];
                tourGroup[tourCount] = tourGroup[topTour];
                tourGroup[topTour] = tempTour;
        }
        bool foundNewBestTour = false;
        childPosition = tourGroup[groupSize - 1];
        population[childPosition] = Tour.Crossover(population[tourGroup[0]], population[tourGroup[1]], cityList, rand);
        if (rand.Next(100) < mutation)
        {
            population[childPosition].Mutate(rand);
        }
        population[childPosition].DetermineFitness(cityList);
        if (population[childPosition].Fitness < population.BestTour.Fitness)
        {   population.BestTour = population[childPosition];
            foundNewBestTour = true;
        }
        childPosition = tourGroup[groupSize - 2];
        population[childPosition] = Tour.Crossover(population[tourGroup[1]], population[tourGroup[0]], cityList, rand);
        if (rand.Next(100) < mutation)
        { population[childPosition].Mutate(rand);
        }
        population[childPosition].DetermineFitness(cityList);
        if (population[childPosition].Fitness < population.BestTour.Fitness)
        {   population.BestTour = population[childPosition];
            foundNewBestTour = true;
        }
        return foundNewBestTour;
    }
    void displayTour(Tour bestTour, int generationNumber, bool complete)
    { if (foundNewBestTour != null)
        { this.foundNewBestTour(this, new TspEventArgs(cityList, bestTour, generationNumber, complete));
        }
    }
```

Şekil 5.Tsp Sınıfı

5.Sonuç

Genetik algoritmalar günümüzde birçok problemin çözümünde kullanılmaktadır. Doğadaki olayların bilgisayar ortamında benzetiminin yapılması ile problem çözüm aşamalarını hızlandıran genetik algoritmalar en iyileme yöntemi ile istenilen sonuçları kullanıcılara sunmaktadır.

Gezgin satıcı probleminin ortaya çıkışı ile probleme genetik algoritmalar tarafından bakılarak yaklaşılmış ve bu gibi karınca kolonisi ya da arı kolonisi uygulamaları gibi şehirler veya noktalar arasında gidilen bir yoldan tekrar geçmemek ve başlanılan noktaya tekrardan gelme olayı genetik algoritmalar kullanılarak bilgisayarlar aracılığı ile çok kısa sürelerde elde edilmiştir.

Uygulama noktasından bakılacak olunursa eğer şehirler arası dolaşımın küçük popülasyonlarda kolay ve hızlı gerçekleşmesi fakat popülasyonun büyümesi ile problemin karmaşıklığı ve içinden çıkılamaz bir hal alması genetik algoritma çözümünü getirmektedir. Bu yüzden etkin çözümler sunabilecek olması ile gezgin satıcı problemi günlük hayatı kolaylaştırması ve farklı yaklaşımlarla çözümler bulması ile bir en iyileme problemi olarak ele alınabilir. Yapılan çalışma ile gezgin satıcı probleminin çözümü ve genetik algoritmaların kullanılması kavramları daha iyi anlaşılmış, yaşanan problemlerin ve gereken iyileştirmelerin nasıl yapılacağı kavranılmıştır.

6.Kaynakça

[1] Çolak, S. Genetik Algoritmalar Yardımı ile Gezgin Satıcı Probleminin Çözümü Üzerine Bir Uygulama, *Çukurova Üniversitesi, Sosyal Bilimler Enstitüsü Dergisi, Cilt:19 Sayı:3 Sayfa :423-438*, Adana, 2010

[2] Traveling Salesman Problem Using Genetic Algorithms (http://www.lalena.com/AI/Tsp/).

[3] Grefenstette, J., Gopal, R., Rosmaita, B., & Van Gucht, D. (1985, July). Genetic algorithms for the traveling salesman problem. In *Proceedings of the first International Conference on Genetic Algorithms and their Applications* (pp. 160-168). Lawrence Erlbaum, New Jersey (160-168).

[4] Grefenstette, J., Gopal, R., Rosmaita, B., & Van Gucht, D. (1985, July). Genetic algorithms for the traveling salesman problem. In *Proceedings of the first International Conference on Genetic Algorithms and their Applications* (pp. 160-168). Lawrence Erlbaum, New Jersey (160-168).