EGE ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
2019 – 2020 ÖĞRETİM YILI

EVRİMSEL HESAPLAMA
PROJE RAPORU

Ege Doğan Dursun - 05170000006
Cem Çorbacıoğlu – 05130000242

Teslim Tarihi : 13 Mayıs 2020

Giriş

1.Projenin amacı nedir?

Projenin amacı, yatay ve dikey akslardan oluşan 2 boyutlu bir düzlem üzerinde tanımlanan şehirlerin içerisine mümkün olan en verimli şekilde baz istasyonlarını yerleştirmektir. Bu noktada, verimliliği belirleyen esas, baz istasyonları tarafından kapsanan müşteri sayısını maksimize etmektir. Şehirlerin, baz istasyonu yerleşim noktalarına sahip olma ya da olmama şeklinde iki tanımı bulunmaktadır. Önceden belirli baz istasyonu yerleşim noktaları bulundurmayan şehirlerde, istasyonların yerleşimi düzlem üzerinde reel sayılar kümesine dahil koordinatlar şeklinde tanımlanmaktayken, önceden belirli baz istasyonu yerleşim noktalarına sahip olan şehirlerde, baz istasyonlarının yerleşimleri bu noktalardan çeşitli kombinasyonlarla seçimler yapılarak sağlanmaktadır.

İki farklı problem tipi için tanımlanan 2 farklı optimizasyon algoritması bulunmaktadır. Önceden belirli yerleşim noktaları arasından seçim yapılmasına yönelik **genetik algoritma** kullanılacakken, belirli yerleşim noktası bulunmayan problem tipine yönelik **sürekli optimizasyon** algoritması kullanılacaktır.

2.Projede kullanılan programlama dili ve IDE

Programlama dili olarak **Java** tercih edilmiştir. Bu dilin tercih edilmesinde derslerde ağırlıklı olarak Java dilinin kullanılması etkili olmuştur. Ek olarak, Python üzerinde de benzer proje geliştirilmiş ve performans olarak Java'nın daha hızlı olması Java'nın tercih edilmesi üzerinde yine etkili olmuştur.

IDE olarak **Eclipse Photon (Haziran 2018 sürümü)** kullanılmıştır.

Kullanılan Kütüphaneler **: Java Collections Kütüphanesi**

3.Ne kadar sürede yapıldı?

Projenin geliştirilmesi için harcanan toplam süre **14 gündür**.
Problem Analizi ve Teknoloji Seçimi : **2 gün**
Kullanılacak Yöntemlerin Belirlenmesi : **1 gün**
Kullanılacak Obje ve Paket Yapılarının Tasarlanması : **1 gün**
Gerekli Optimizasyon Metotları ve Obje Sınıflarının Kodlanması : **4 gün**
Yardımcı Sınıfların Kodlanması : **1 gün**
Makro Sınıfların Kodlanması : **1 gün**
Test Sınıflarının Kodlanması : **2 gün**
Parametre Optimizasyonu ve Testler : **1 gün**
Rapor Yazımı : **1 gün**

İçerik

1.Bölümlerde İstenen Ana Metotların Açıklaması

(BÖLÜM -1)

```java
package tests;

import model.City;

public class GlobalGAVariables {

    /*
     *  THESE VARIABLES DEFINE THE "CITY A" THAT WAS MENTIONED IN THE HOMEWORK (PART : 2)
     *
     * */

    //Problem parameters
    private static double width = 400;
    private static double height = 400;
    private static int customerAmount = 250;
    private static int baseLocationAmount = 100;
    private static double coverRadius = 30;
    private static int baseAmount = 50;

    //Defining the city and the problem for CITY A
    public static City cityA = new City(width, height, customerAmount, baseLocationAmount, baseAmount);
    public static Problem problemA = new Problem(cityA, coverRadius);
}
```

```java
package tests;

import model.City;

public class GlobalDEVariables {

    /*
     *  THESE VARIABLES DEFINE THE "CITY B" THAT WAS MENTIONED IN THE HOMEWORK (PART : 3)
     *
     * */


    //Problem parameters
    private static double width = 500;
    private static double height = 500;
    private static int customerAmount = 300;
    private static int baseLocationAmount = 0;
    private static double coverRadius = 35;
    private static int baseAmount = 50;

    //Defining the City and Problem Objects for CITY B
    public static City cityB = new City(width, height, customerAmount, baseLocationAmount, baseAmount);
    public static Problem problemB = new Problem(cityB, coverRadius);

}
```

Test sınıfı içerisindeki GlobalGAVariables sınıfının içerisinde genetik algoritma ve sürekli optimizasyon problemlerimizle ilgili temel parametreleri tanımlıyoruz. Ardından ilgili parametreleri kullanarak şehir A ve şehir B için objelerimizi ve Problem instance'larımızı yaratıyoruz. Burada dikkat edilmesi gereken nokta, B şehri için; yani sürekli optimizasyon gerçekleştirilecek olan şehrimizdeki baz istasyonu yerleşim noktalarının sayısını ifade eden *baseLocationAmount* değişkeninin sıfır değer almasıdır. Bu sıfır değeri tanımlaması Problem nesnemizin constructor'ının problemin bir sürekli optimizasyon problemi olduğunu anlamasını sağlamaktadır.

```java
    private int id;
    private double width;
    private double height;
    private int baseLocationAmount;
    private int customerAmount;
    private int baseAmount;

    private static int counter = 0;

    public City(double width, double height, int customerAmount, int baseLocationAmount, int baseAmount) {
        counter++;
        this.id = counter;
        this.width = width;
        this.height = height;
        this.customerAmount = customerAmount;
        this.baseLocationAmount = baseLocationAmount;
        this.baseAmount = baseAmount;
    }

    public City(double width, double height, int customerAmount, int baseAmount) {
        counter++;
        this.id = counter;
        this.width = width;
        this.height = height;
        this.customerAmount = customerAmount;
        this.baseLocationAmount = 0;
        this.baseAmount = baseAmount;
    }
```

Şehir objesinin tanımında görüldüğü üzere, şehirlerin baz istasyon yerleşim noktalarının varlığını ve yokluğunu ayrı koşullar olarak değerlendiren iki farklı constructor'ı bulunmaktadır. Bu sayede sürekli ve sürekli olmayan optimizasyonları aynı anda işleyebilecek ortak bir tanım nesnesi meydana getirebilmiş oluyoruz.

```java
public class Problem {

    /*
     *
     * PROBLEM IS THE OBJECT THAT ACTS AS A WRAPPER OF THE PROBLEM DEFINITION FOR OUR OPTIMIZATION PROBLEMS
     * IT CAN WORK WITH DIFFERENT RANDOMIZED SETS OF PARAMETERS AND IT CAN ALSO WORK ON DIFFERENT CITIES WITH DIFFERENT PARAMETERS,
     * DIFFERENT COVER RADIUS -> WHICH IS THE MAXIMUM DISTANCE A BASE CAN SERVE.
     *
     */

    private City city;
    private ArrayList<Customer> customers;
    private ArrayList<BaseLocation> baseLocations;
    private double coverRadius;


    public Problem(City city, double coverRadius) {
        this.city = city;
        this.customers = randomizeCustomers();
        System.out.println("\n PROBLEM - CUSTOMERS AND BASE LOCATIONS : ");

        //Show customer coordinates on problem creation
        showCustomerCoordinates();
        this.baseLocations = randomizeBaseLocations();

        //Show base location coordinates on problem creation (if exists)
        showBaseLocationCoordinates();
        this.coverRadius = coverRadius;
    }
```

Problem objemizin sınıfı içerisinde ise ödevin **Bölüm-1** kısmında bahsi geçen düzlem içerisinde randomize baz istasyonu yerleşim noktaları belirleme, randomize müşteri noktaları belirleme ve ardından bu noktaları konsolda listeleme gibi işlevleri gerçekleştiren metotlar yer almaktadır.

```java
//This method randomizes customer objects and their locations on the specified city.
public ArrayList<Customer> randomizeCustomers(){

    Random r = new Random();
    ArrayList<Customer> customers = new ArrayList<Customer>();
    double x;
    double y;
    for(int i =0; i<city.getCustomerAmount(); i++) {
        x = city.getWidth() * r.nextDouble();
        y = city.getHeight() * r.nextDouble();
        Customer customer = new Customer(x, y);
        customers.add(customer);
    }

    return customers;
}
```

Müşterilerin düzlem üzerindeki randomize noktalara yerleştirilmesi işlevini üstlenen metot yukarıdaki gibidir. Bu metot sayesinde, müşterileri yerleştirmek için düzlem üzerinde double veri tipinde rasgele x ve y değişkenlerinin belirlenmesi sağlanır ve ardından bu özelliklere sahip müşteri nesneleri meydana getirilir. Bu müşteri nesneleri problem objesinin ilgili özelliğine atanır.

```java
package model;

public class Customer {

    /*
     *
     * CUSTOMERS ARE REPRESENTING THE MAJOR FACTOR THAT AFFECTS THE FITNESS AS THE AIM IS TO OPTIMIZE THE NUMBER OF CUSTOMERS
     * WHO CAN ACHIEVE SIGNALS FROM THE TELECOMMUNICATION BASES.
     *
     * EACH CUSTOMER HAS SPECIFIC COORDINATES IN THE CITY / 2-DIMENSIONAL SPACE SUCH THAT X -> HORIZONTAL AXIS & Y -> VERTICAL AXIS
     *
     *
     */

    private int id;
    private double x;
    private double y;

    private static int counter = 0;

    public Customer(double x, double y) {
        counter++;
        this.id = counter;
        this.x = x;
        this.y = y;
    }
```

Müşteriler, instance olarak yaratım sıralarına bağlı birer id'ye ek olarak düzlemdeki yer alış noktalarını belirleyen bir x (yatay) ve y (dikey) koordinata sahiptirler. Koordinatlar double veri tipinde tanımlanmıştır.

```java
//this method randomizes base location objects and their locations on the specified city.
public ArrayList<BaseLocation> randomizeBaseLocations(){

    if(city.getBaseLocationAmount() == 0) {
        return null;
    }
    else {

        Random r = new Random();
        ArrayList<BaseLocation> baseLocations = new ArrayList<BaseLocation>();
        double x;
        double y;
        for(int i = 0; i<city.getBaseLocationAmount(); i++) {
            x = city.getWidth() * r.nextDouble();
            y = city.getHeight() * r.nextDouble();
            BaseLocation baseLocation = new BaseLocation(x, y);
            baseLocations.add(baseLocation);
        }

        return baseLocations;
    }

}
```

Problem sınıfına ait olan bir diğer metot olan *randomizeBaseLocations* ise, esas olarak problemin tanımında belirtilen sayıda baz istasyonu yerleşim noktasının düzlem üzerinde rasgele yerleştirilmesi görevini üstlenen metottur. double veri tipinde rasgele üretilen yatay ve dikey koordinatlar baz istasyonu yerleşim noktası objesi meydana getirmek için kullanılır ve üretilen baz istasyonu yerleşim noktaları ilgili Problem nesnesine atanır.

```java
package model;

public class BaseLocation {

    /*
     *
     * THIS CLASS DEFINES THE BASE LOCATION OBJECT WHICH IS REPRESENTING THE POSSIBLE LOCATIONS INSIDE A CITY
     * THAT THE TELECOMMUNICATIONS COMPANY CAN USE TO PUT THE BASE OBJECTS.
     *
     * EACH BASELOCATION HAS 2-DIMENSIONAL COORDINATES IN THE CITY SUCH THAT X -> HORIZONTAL & Y -> VERTICAL AXIS.
     *
     * EACH BASELOCATION ALSO HAS AN ASPECT CALLED isOccupied, SO WHILE RANDOMIZING THE BASES, THERE CAN'T BE MORE THAN 1 BASES AT THE SAME BASE
     *
     */

    private int id;
    private double x;
    private double y;
    private boolean isOccupied;

    private static int counter = 0;

    public BaseLocation(double x, double y) {
        counter++;
        this.id = counter;
        this.x = x;
        this.y = y;
        this.isOccupied = false;
    }
```

Baz istasyonu yerleşim noktası objeleri, temel olarak instance olarak yaratım sıralarını dikkate alarak belirlenen bir id'ye sahip olmakla birlikte 2 boyutlu koordinat sistemi üzerindeki konumlarını belirleyen x(yatay) ve y(dikey) koordinat değişkenlerine sahiptir. Bunun dışında herhangi bir baz istasyonu tarafından doldurulup doldurulmadıklarını kontrol etme amaçlı olarak *isOccupied* değişkenine sahiptirler. Bunun temel sebebi, bir baz istasyonunun bir yerleşim noktasını birden kez işgal etmesini önlemektir.

```java
//This method makes us be able to see the coordinates of the randomized customers on creation
public void showCustomerCoordinates() {

    System.out.println("\n CUSTOMER LOCATIONS (X / Y) : ");
    for(int i = 0; i<this.customers.size(); i++) {
        System.out.println("Customer "+i+" ("+
                           String.format("%.2f", customers.get(i).getX())+
                           " / "+
                           String.format("%.2f", customers.get(i).getY())+
                           ") , ");
    }
    System.out.println();
}

//This method makes us able to see the coordinates of the randomized base locations on creation (if exists)
public void showBaseLocationCoordinates() {

    if(city.getBaseLocationAmount() != 0) {
        System.out.println("\n BASE LOCATION COORDINATES (X / Y) : ");
        for(int i = 0; i<baseLocations.size(); i++) {
            System.out.println("Base Location "+i+" ("+
                    String.format("%.2f", baseLocations.get(i).getX())+
                    " / "+
                    String.format("%.2f", baseLocations.get(i).getY())+
                    ") , ");
        }
        System.out.println();
    }
    else {

        //If there is no base location, inform the user that the system is continuous.
        System.out.println("\n This city does not have mandatory base locations! The problem is continuous!");
        System.out.println();
    }
}
```

Ödevin **Bölüm-1** kısmında gerçekleştirilmesi istenen; konsola rasgele üretilen müşteri ve baz istasyonu yerleşim noktaları ile ilgili bilgilerin yerleştirilmesi işlevini bu metotlar üstlenmektedir. Problem sınıfına atanan müşteriler ve baz istasyonu yerleşim noktaları listeleri boyutlarınca taranarak sahip oldukları koordinatlar konsolda listelenir. Constructor yapısı gereği Problem objeleri, tanımlandıkları ve yaratıldıkları anda sahip oldukları randomize koordinatlar ile ilgili bilgileri otomatik olarak konsolda gösterecektir.

```java

package tests;

import model.Problem;

public class ProblemDefinitionsTest {

    /*
     *
     * THIS CLASS IS USED FOR TESTING THE PRINTING OF THE PROBLEM DEFINITIONS WHICH WAS MENTIONED ON HOMEWORK -> PART 1
     *
     */

    public static void main(String[] args) {

        //Achieving PROBLEM A -> solves the problem in CITY A
        Problem problemA = GlobalGAVariables.problemA;
        problemA.toString();

        //Achieving PROBLEM B -> solved the problem in CITY B
        Problem problemB = GlobalDEVariables.problemB;
        problemB.toString();

    }
}

```

ProblemDefinitionsTest sınıfının içerisinde ödevin **Bölüm-1** kısmının işlevi test edilmiştir.

(BÖLÜM – 2)

```
public class Individual implements Comparable<Individual> {

    /*
     *
     * THIS CLASS IS USED FOR DEFINING THE INDIVIDUAL OBJECT, WHICH IS THE MAIN POPULATION ACTOR IN THE GENETIC ALGORITHM
     *
     * IT CONTAINS A PROBLEM OBJECT, WHICH IS DEFINING THE PROBLEM TO BE SOLVED, A FITNESS VARIABLE WHICH DEFINES FITNESS LEVEL,
     * AND A CHROMOSOME THAT CONTAINS THE BASE COORDINATES THE INDIVIDUAL POSSESS.
     *
     */


    private int id;
    private Problem problem;
    private ArrayList<Base> chromosome;
    private Double fitness;

    private static int counter = 0;

    public Individual(Problem problem) {
        counter++;
        this.id = counter;
        this.problem = problem;
        this.chromosome = randomizeChromosome();
        this.fitness = 0.0;
    }
```

Ödevin **Bölüm-2** kısmında gerçekleştirilmesi istenen genetik algoritmayı modellemek için üretilen birey sınıfı yukarıda tanımlandığı özelliklere sahiptir. Birey sınıfı, instance yaratım sıralarına bağlı bir id'ye ek olarak, bağlı oldukları problemi içeren bir problem nesnesi değişkenine ve bu probleme getirdikleri çözümü ifade eden (kromozom) bir *chromosome* değişkenine sahiptirler. Bu değişken özünde baz istasyonu nesnelerinden oluşan bir ArrayList'dir. Elbette, popülasyon içerisindeki değerlerinin kayıtta tutulabilmesi için, bireyler bir uygunluk (fitness) değerine de sahiptir.

```
package model;

public class Base {

    /*
     *
     * THIS CLASS DEFINES THE BASE OBJECT THAT WILL BE USED FOR DESCRIBING THE GENE IN THE CHROMOSOME OF THE INDIVIDUAL
     *
     * EACH BASE HAS A LOCATION IN THE 2-DIMENSIONAL SPACE SUCH THAT X -> HORIZONTAL  &  Y -> VERTICAL AXIS.
     *
     * EACH BASE ALSO HAS A COVER RADIUS WHICH MEANS HOW FAR THEY CAN SERVE THE TELECOMMUNICATION SERVICES AT MAXIMUM.
     *
     */

    private int id;
    private double x;
    private double y;
    private double coverRadius;


    private static int counter = 0;


    public Base(double x, double y, double coverRadius) {
        counter ++;
        this.id = counter;
        this.x = x;
        this.y = y;
        this.coverRadius = coverRadius;
    }
```

Baz istasyonu nesneleri, düzlem üzerine direkt olarak yerleştirilebilen; veya önceden belirlenmiş baz istasyonu yerleşim noktaları üzerine yerleştirilebilen nesneleri temsil etmektedir. Baz istasyonlarının temel olarak düzlem üzerindeki konumlarını ifade eden x(yatay) ve y(dikey) koordinat değişkenleri bulunmaktadır. Buna ek olarak ne kadar uzaktaki müşterileri kapsayabildiklerini ifade eden *coverRadius* değişkenleri bulunmaktadır. Bu değişken yaratımları sırasında Problem sınıfına bağlı olarak şekillenmektedir.

```java
//This method is used for randomizing the chromosome of the individual on creation
public ArrayList<Base> randomizeChromosome(){

    Random r = new Random();
    double x;
    double y;
    ArrayList<Base> bases = new ArrayList<Base>();
    if(problem.getCity().getBaseLocationAmount() == 0) {

        for(int i = 0; i<problem.getCity().getBaseAmount(); i++) {

            //create random coordinates in continuous space if there are no base locations
            x = problem.getCity().getWidth() * r.nextDouble();
            y = problem.getCity().getHeight() * r.nextDouble();
            Base base = new Base(x, y, problem.getCoverRadius());
            bases.add(base);
        }

        return bases;
    }
    else {

        for(BaseLocation location  : problem.getBaseLocations()) {
            location.setOccupied(false);
        }

        int choice;
        for(int i = 0; i<problem.getCity().getBaseAmount(); i++) {

            choice = r.nextInt(problem.getBaseLocations().size());
            if(problem.getBaseLocations().get(choice).isOccupied() == true) {
                i = i -1;
                continue;
            }
            else {

                //select from baselocations if they are defined.
                x = problem.getBaseLocations().get(choice).getX();
                y = problem.getBaseLocations().get(choice).getY();
                Base base = new Base(x, y, problem.getCoverRadius());
                bases.add(base);
                problem.getBaseLocations().get(choice).setOccupied(true);
            }
        }

        return bases;
    }
}
```

Bireylerin yaratım sırasında rasgele kromozom yapılarının üretilmesini sağlayan metot yukarıda belirtilmiştir. Kontrol yapılarından da anlaşılacağı üzere, eğer bir bireyin bağlı olduğu problemin önceden belirli baz istasyonlarını içeren bir tanımı varsa, randomize baz istasyonu lokasyonlarının belirlenmesinde bu lokasyonlar göz önünde bulundurulacaktır. İlgili yerleşim lokasyonları arasından seçim yapıldıktan sonra bu rasgele seçimler Bireylerin baz istasyonlarını içeren kromozom değişkenine kaydedilecektir. Aynı şekilde baz istasyonlarının kapsama uzaklıklarının belirlenmesinde problem nesnesinin sahip olduğu tanım değerleri göz önüne alınmaktadır.

```java
//This method is used for showing the chromosome values of the individual
public void showChromosome() {
    System.out.println();
    System.out.println("\n INDIVIDUAL BASE COORDINATES : ");
    for(int i = 0; i<chromosome.size(); i++) {
        Base base = chromosome.get(i);
        System.out.print("Base "+i+" ("+
                         String.format("%.2f", base.getX())+
                         " / "+
                         String.format("%.2f", base.getY())+
                         ") , ");
    }
    System.out.println();
}
```

Bir bireyin kromozom yapısının gözlenebilmesi ve konsolda görüntülenebilmesi adına yazılmış bir metottur. Kromozomun içeriğinde yer alan baz istasyonlarının koordinatlarını listeler.

```
package genetic_algorithm;

import java.util.ArrayList;

public class Population {


    /*
     *
     * THIS CLASS IS USED FOR GENERATING POPULATIONS FOR GENETIC ALGORITHMS WITH SPECIFIED PARAMETERS
     *
     */


    //This method is used for generating a population for the genetic algorithm
    public static ArrayList<Individual> generatePopulation(Problem problem, int populationSize){

        ArrayList<Individual> population = new ArrayList<Individual>();
        for(int i = 0; i<populationSize; i++) {
            Individual individual = new Individual(problem);
            population.add(individual);
        }

        return population;
    }
```

Popülasyon sınıfı genetik algoritmanın popülasyonla bağlantılı faaliyetlerini yürütmesi için gerekli statik metotları içermektedir. Yukarıda görüntülenen metot, popülasyon yaratma aşamasını gerçekleştirmektedir. Popülasyonu yaratmak için ilgili Birey nesnelerinden belirlenen popülasyon boyutunda olacak şekilde bir ArrayList yaratma görevini üstlenir. Elde edilen popülasyonu geri döndürür.

```
  //This method is used for showing the basic data about the population including fitness scores.
  public static void showPopulationData(ArrayList<Individual> population) {

      System.out.println();
      System.out.println("PRINTING THE INFORMATION OF THE POPULATION ... ");
      System.out.println();
      for(int i = 0; i<population.size(); i++) {
          Individual iv = population.get(i);
          System.out.println("INDIVIDUAL NUMBER: " + i);
          System.out.println("INDIVIDUAL FITNESS SCORE: " + iv.getFitness());
          iv.showChromosome();
          System.out.println("_________________");
      }
  }
```

Bu metotun görevi, popülasyondaki her bireyle ilgili bilgilendirmeyi konsolda listelemektir. Bireyler ile ilgili verilen ana bilgi, uygunluk değerleridir.

```java
    //This method is used for combining the different population groups in the ga; such as the elites, elders, children and random individua
    public static ArrayList<Individual> combinePopulation(ArrayList<Individual> elders, ArrayList<Individual> children, ArrayList<Individual>

        ArrayList<Individual> combinedPopulation = new ArrayList<Individual>();

        for(int i = 0; i<elders.size(); i++) {
            combinedPopulation.add(elders.get(i));
        }

        for(int i = 0; i<children.size(); i++) {
            combinedPopulation.add(children.get(i));
        }

        for(int i = 0; i<randoms.size(); i++) {
            combinedPopulation.add(randoms.get(i));
        }

        //return the combined population
        return combinedPopulation;
    }
}
```


Bu metotun görevi ise, genetik algoritmayı yürütme aşamasında gerçekleştirilen, çocuk birey üretme, elit bireyleri gelecek nesle aktarma, rasgele bireyler seçme gibi çeşitli metotlar sonucu üretilen alt-popülasyonların birleştirilerek gelecek nesle aktarılacak yeni bir birleşik popülasyon meydana getirilmesidir.

```java
package genetic_algorithm;

import java.util.ArrayList;

public class Fitness {


    /*
     *
     * THIS CLASS IS USED FOR FITNESS EVALUATION AND REPRESENTATION OF THE GENETIC ALGORITHM POPULATION
     *
     * IT CONTAINS THE METHODS THAT ARE RELATED WITH FITNESS EVALUATION AND REPRESENTATION
     *
     */

    //This method is used for evaluating the fitness score of an individual in genetic algorithm population
    public static double evaluateIndividualFitness(Problem problem, Individual iv) {

        ArrayList<Customer> customers = problem.getCustomers();

        double fitness = 0;
        for(int j = 0; j<customers.size(); j++) {

            Customer customer = customers.get(j);
            for(int k = 0; k<iv.getChromosome().size(); k++) {

                Base base = iv.getChromosome().get(k);

                //calculating the distance of a customer and checking if he/she is covered by a base
                double deltaXPow = Math.pow(customer.getX() - base.getX(), 2);
                double deltaYPow = Math.pow(customer.getY() - base.getY(), 2);
                double sqrtSum = Math.sqrt((deltaXPow + deltaYPow));


                //increate the fitness score if he/she is covered
                if(sqrtSum <= problem.getCoverRadius()){
                    fitness = fitness + 1;
                    break;
                }else {
                    continue;
                }
            }
        }

        //set the fitness level of the individual
        iv.setFitness(fitness);

        return fitness;
    }
```

Fitness sınıfı genetik algoritmanın yürütülebilmesi ve sonraki nesle yönelik daha uygun olan bireylerin seçilebilmesi için gerekli uygunluk hesaplama yöntemlerini içermektedir. Yukarıda görüntülenen metot ise, popülasyona ait tek bir bireyin uygunluk değerini ölçümlemek ve sınıfındaki öz değerine atamak amacıyla dizayn edilmiştir. Uygunluk değerinin ölçümlenmesinde temel faktör, sahip olduğu özgün koordinatlı baz istasyonlarının müşterilerin ne kadarını kapsadığının ölçümlenmesidir. Bunun için uzaklık hesaplaması kullanılmaktadır.

```java
//This method is used for evaluating the fitness level of a whole genetic algorithm population. It is a wrapper function that uses the
public static ArrayList<Individual> evaluatePopulationFitness(Problem problem, ArrayList<Individual> population){

    for(int i = 0; i<population.size(); i++) {

        Individual iv = population.get(i);


        //evaluate and get the fitness level of the individual and set it
        double fitness = evaluateIndividualFitness(problem, iv);

        iv.setFitness(fitness);
    }

    return population;
}
```

Bu metot ise basit şekilde birey uygunluk hesap fonksiyonun paketleme fonksiyonudur ve bir popülasyon içerisindeki tüm bireylerin uygunluk değerlerini hesaplamayı amaç edinmektedir. Uygunluk değerleri hesaplanan popülasyon geri döndürülür.

```java
//This method is used for showing the fitness scores of the individuals in a population one by one
public static void showPopulationFitnessScores(Problem problem, ArrayList<Individual> population) {

    System.out.println();
    System.out.println("SHOWING POPULATION FITNESS SCORES ...");
    System.out.println();
    for(int i = 0; i<population.size(); i++) {
        double prc = (population.get(i).getFitness()/problem.getCity().getCustomerAmount())*100;
        System.out.println("INDIVIDUAL "+i+" FITNESS SCORE : " + population.get(i).getFitness() + " / "+ problem.getCity().getCustomerAmount() + " ( %" + prc + " )
    }
    System.out.println();
}


//This method is used for getting the average population score in a population
public static double getAveragePopulationFitness(ArrayList<Individual> population) {

    double summation = 0;
    for(int i = 0; i<population.size(); i++) {

        summation = summation + population.get(i).getFitness();
    }

    double average = (double) (summation/population.size());

    return average;
}


//This method is used for printing the average population score in a genetic algorithm population
public static void showAveragePopulationFitness(Problem problem, ArrayList<Individual> population) {

    double summation = 0;
    for(int i = 0; i<population.size(); i++) {

        summation = summation + population.get(i).getFitness();
    }

    double average = summation/population.size();

    double prc = (average/problem.getCity().getCustomerAmount())*100;
    System.out.println("AVERAGE FITNESS OF POPULATION : "+average+" / "+problem.getCity().getCustomerAmount() + " ( %" + prc + " )");
}
}
```

Bu üç metotun ise gerçekleştirdiği temel işlevler, popülasyona ait her bireyin uygunluk değerlerini listelemek, popülasyonun ortalama uygunluk değerini geri döndürmek ve popülasyonun ortalama uygunluk değerini konsolda görüntülemektir. Fitness sınıfı için bunların yardımcı metotlar olduğunu söyleyebiliriz.

```java
package genetic_algorithm;

import java.util.ArrayList;

public class Selection {


    /*
     *
     * THIS CLASS IS USED FOR DEFINING THE SELECTION METHODS FOR THE GENETIC ALGORITHM
     *
     */


    //this method is used for selecting elite individuals from the population according to their fitness level.
    public static ArrayList<Individual> selectElitesByFitness(ArrayList<Individual> population,
                                                              double elitesAmount){

        //define the elites group and the sorted population (by fitness score)
        ArrayList<Individual> elites = new ArrayList<Individual>();
        ArrayList<Individual> sortedPopulation = sortPopulationByFitness(population);

        for(int i = 0; i<elitesAmount; i++) {

            elites.add(sortedPopulation.get(i));
        }

        return elites;
    }
```

Bir sonraki önemli nokta ise, uygunluk değerleri belirlenmiş popülasyon bireyleri içerisinden en yüksek uygunluğa sahip elit bireylerin seçilmesidir. Bu seçim işleminin gerçekleştirilmesi için Selection sınıfındaki statik metotlar kullanılmaktadır. Yukarıda görüntülenen metotun amacı, elit bireylerin seçiminin gerçekleşmesi adına öncelikle elde edilen popülasyonu fitness değerine göre dizimletmek ve ardından parametre olarak belirtilen elit birey sayısı kadar en iyisini seçerek bu bireyleri geri döndürmektir.

```java
  //this method is used for sorting the population arraylist by fitness score.
  public static ArrayList<Individual> sortPopulationByFitness(ArrayList<Individual> population){

      Collections.sort(population);
      Collections.reverse(population);

      return population;
  }
```

Popülasyondaki bireylerin uygunluk değerlerine bağlı olarak sıralı şekle sokulmasında Collections kütüphanesinden faydalanılmıştır.

```java
//this method is used for selecting successful elders that will pass to other generation by their fitness score.
public static ArrayList<Individual> selectElders(ArrayList<Individual> population,
                            ArrayList<Individual> elites,
                            double eldersAmount) {

    ArrayList<Individual> elders = new ArrayList<Individual>();

    for(int i = 0; i<eldersAmount; i++) {

        elders.add(elites.get(i));
    }

    return elders;
}
```

Yukarıdaki metot ise, (eğer istenirse) bir sonraki nesile aktarılmak üzere elitlerin arasından belirli yaşlı bireylerin seçilmesini sağlamaktadır. Seçilen bireyler geri döndürülür.

```java
//this method is used for randomly selecting individuals from the population to pass to next generation
public static ArrayList<Individual> randomlySelectIndividuals(ArrayList<Individual> population,
                                                              double randomAmount){

    ArrayList<Individual> randomIvs = new ArrayList<Individual>();

    Random r = new Random();
    for(int i = 0; i<randomAmount; i++) {

        int choice = r.nextInt(population.size());
        randomIvs.add(population.get(choice));
    }

    return randomIvs;
}
```

Bu metot da aynı şekilde, eğer istenirse bir sonraki nesile mevcut jenerasyon içinden rasgele bireylerin seçilerek aktarılmasını sağlamaktadır. Parametre ile kısıtlanan oranda birey seçimi gerçekleştirilmektedir.

```java
package genetic_algorithm;

import java.util.ArrayList;

public class Mating {

    /*
     *
     * THIS CLASS IS FOR DEFINING THE METHODS THAT ARE USED FOR GENETIC ALGORITHM INDIVIDUAL AND POPULATION BREEDING FUNCTIONS
     *
     */


    //This method is used for breeding a population and applying crossover in a given rate. It is actually a wrapper function that uses individual breeding function.
    public static ArrayList<Individual> breedPopulation(Problem problem, ArrayList<Individual> population, double amountChildren, double crossoverRate) {

        Random r = new Random();
        ArrayList<Individual> children = new ArrayList<Individual>();
        Individual child = new Individual(problem);

        for(int i = 0; i<amountChildren; i++) {

            //select parents from the population
            int parent1Index = r.nextInt(population.size());
            int parent2Index = r.nextInt(population.size());

            Individual parent1 = population.get(parent1Index);
            Individual parent2 = population.get(parent2Index);

            //produce the children
            child = produceChildrenByCrossover(problem, parent1, parent2, crossoverRate);
            children.add(child);
        }

        return children;
    }
```

Ardından seçilen elit bireylerin, çocuk bireyleri oluşturmak adına çiftleştirilmesi görevini üstlenen metotları içeren Mating sınıfı kullanılacaktır. Yukarıda görüntülenen metodun görevi, elit bireyler veya popülasyon şeklinde verilen herhangi bir birey nesnesi listesi içerisindeki her bireyi, belirtilen çaprazlama oranlarını göz önünde bulundurarak çiftleştirmek ve istenen çocuk sayısını aşmayacak şekilde çocuklar meydana getirmektedir. Bir anne ve baba çiftinin çocuk oluşturulması için kullanılan metotun popülasyon boyutunda paketleyicisidir.

```java
  //This method is used for producing children from parent individuals and applies crossover in the given rate.
  public static Individual produceChildrenByCrossover(Problem problem, Individual parent1, Individual parent2, double crossoverRate) {

      Random r = new Random();
      int coDie = r.nextInt(100);
      Individual child = new Individual(problem);
      ArrayList<Base> chromosome = new ArrayList<Base>();
      crossoverRate = crossoverRate*100;

      //apply crossover if the die is lower than the rate
      if(coDie < crossoverRate) {

          int coIndex = r.nextInt(parent1.getChromosome().size());

          //split and merge the chromosomes of the parents
          for(int i = 0; i<coIndex; i++) {
              chromosome.add(parent1.getChromosome().get(i));
          }

          for(int i =coIndex; i<parent1.getChromosome().size(); i++) {
              chromosome.add(parent2.getChromosome().get(i));
          }

          child.setChromosome(chromosome);

      }
      else {
          int choice = r.nextInt(2);

          //dont apply crossover and select a random parents chromosome
          if(choice == 1) {
              chromosome = parent1.getChromosome();
          }
          else {
              chromosome = parent2.getChromosome();
          }

          child.setChromosome(chromosome);

      }

      return child;
  }
```

Bu metot, anne ve baba bireyler, belirtilen çaprazlama oranları dahilinde işleme tabi tutarak bir çocuk üretilmesini sağlayan metottur. Çaprazlama dahilinde anne ve babanın kromozomlarının belirli bir rasgele noktadan kesilmesi ve değiştirilerek birleştirilmesi sağlanır.

```java
package genetic_algorithm;

import java.util.ArrayList;

public class Mutation {

    /*
     *
     * THIS CLASS IS USED FOR DEFINING THE METHODS FOR APPLYING MUTATION TO THE GENETIC ALGORITHM INDIVIDUALS AND POPULATION
     *
     */


    //this method is used for mutating the whole genetic algorithm population and acts as a wrapper function that reaches the individual mutation function
    public static ArrayList<Individual> mutatePopulation(Problem problem, ArrayList<Individual> population, double mutationRate) {

        ArrayList<Individual> mutatedPopulation = new ArrayList<Individual>();
        for (int i = 0; i<population.size(); i++) {

            //mutate each individual
            Individual mutatedIv = mutateIndividual(problem, population.get(i), mutationRate);
            mutatedPopulation.add(mutatedIv);
        }

        return mutatedPopulation;
    }
```

Son olarak popülasyonun belirtilen oranda rasgele mutasyona tabi tutulması amacıyla tasarlanmış metotları içeren Mutation sınıfından söz edeceğiz. Yukarıda belirtilen metot, bireyin mutasyona uğratılmasını sağlayan metotun paketleyici metotu olup; popülasyonun belirtilen oranda mutasyona uğratılmasını sağlamaktadır. Üretilen mutasyona uğramış topluluk geri döndürülür.

```java
//this method is used for mutating a single individual with a given mutation rate.
public static Individual mutateIndividual(Problem problem, Individual iv, double mutationRate) {

    Random r = new Random();
    int muteDie = r.nextInt(100);
    Individual mutatedIv = new Individual(problem);
    mutationRate = mutationRate*100;

    //if the mutation die is smaller than the mutation rate, apply mutation; otherwise leave as it is.
    if(muteDie < mutationRate) {

        int mutationIndex = r.nextInt(iv.getChromosome().size());
        ArrayList<Base> mutatedChromosome = mutateGene(problem, iv.getChromosome(), mutationIndex);
        mutatedIv.setChromosome(mutatedChromosome);

    }
    else {

        mutatedIv = iv;

    }

    return mutatedIv;
}
```

Yukarıdaki metot, bir bireyin belirtilen oranda mutasyona uğratılmasını kontrol eden fonksiyondur. Bu noktada bireyin mutasyona uğratılması için rasgele bir mutasyon endeksi seçilmektedir. Ardından ilgili genin mutasyona uğratılmasını sağlayan *mutateGene* fonksiyonu çağrılmaktadır.

```java
//this method is used for mutating a single gene in a chromosome.
public static ArrayList<Base> mutateGene(Problem problem, ArrayList<Base> chromosome, int mutationIndex) {

    Random r = new Random();
    double x;
    double y;
    double coverRadius = problem.getCoverRadius();
    Base base;

    //apply mutation according to the problem structure.
    if(problem.getCity().getBaseLocationAmount() == 0) {

        x = problem.getCity().getWidth() * r.nextDouble();
        y = problem.getCity().getHeight() * r.nextDouble();
        base = new Base(x, y, coverRadius);
    }
    else {

        ArrayList<BaseLocation> baseLocations = problem.getBaseLocations();
        int baseChoice = r.nextInt(baseLocations.size());
        BaseLocation baseLoc = baseLocations.get(baseChoice);

        x = baseLoc.getX();
        y = baseLoc.getY();
        base = new Base(x, y, coverRadius);
    }

    chromosome.set(mutationIndex, base);
    return chromosome;
}
```

Gen mutasyonu fonksiyonu, problem tanımından bilineceği üzere baz istasyonu yerleşim noktalarının bulunup bulunmayışı faktörünü göz önünde bulundurarak mevcut baz istasyonları arasından seçilen endekste, eskisinden daha farklı bir koordinat seçimi sağlama yoluyla bir geni mutasyona uğratmış olur. Mutasyona uğratılan kromozom geri döndürülür.

```java
package tests;

import java.util.ArrayList;

public class GAGeneralTest {

    public static void main(String[] args) {


        /*
         *  THIS CLASS IS USED FOR TESTING THE USE OF A CONTINUOUS OPTIMIZATION ALGORITHM FOR OPTIMIZING
         *  THE PROBLEM THAT WAS MENTIONED IN "HOMEWORK PART 2 : CITY A"
         *
         *  USED METHOD : GENETIC ALGORITHM
         *
         *  SELECTION WORKS AS THE FOLLOWING :
         *
         *  -> AN ELITE GROUP IS SELECTED AMONG THE POPULATION EACH EPOCH, REGARDING THEIR FITNESS SCORES.
         *  -> A SPECIFIC PERCENT OF THIS ELITE GROUP TRANSFERS TO THE NEXT GENERATION DIRECTLY, AS ELDERS.
         *  -> BY USING THE ELITE GROUP'S GENES, SPECIFIC PERCENTAGE OF CHILDREN ARE CREATED, AS WELL AS BEING ABLE TO APPLY A CROSSING-OVER; PASSING TO OTHER GENEI
         *  -> A SPECIFIC PERCENT OF RANDOM INDIVIDUALS CAN BE SELECTED FROM THE POPULATION TO PASS TO OTHER GENERATION
         *  -> MUTATION CAN APPLY, CONTROLLED BY THE MUTATION RATE
         *
         *  IN THIS CLASS, YOU CAN EVALUATE THE PERFORMANCE OF THE GENETIC ALGORITHM
         *
         */

        //Genetic Algorithm Hyper-parameters
        int populationSize = 100;
        int maxGenerations = 2000;
        double elitismAmount = (populationSize*0.4);
        double elderAmount = (populationSize*0*1);
        double childrenAmount = (populationSize*0.89);
        double randomIndividualAmount = (populationSize*0.01);
        double crossoverRate = 0.9;
        double mutationRate = 0.1;


        //Multi-Solution Hyper-parameters
        double parameterRandomizationTimes = 4;
        double problemSolutionTimes = 25;


        //Define the city and the problem
        Problem problemA = GlobalGAVariables.problemA;
```

Yukarıdaki genetik algoritma yoluyla gerçekleştirilecek optimizasyonun test sınıfını görmekteyiz. Genetik algoritma için kullanılan hiper parametreleri kısaca tanımlamak gerekirse;

- *populationSize* : Popülasyonda yer alacak birey sayısı
- *maxGenerations* : Genetik algoritmanın üzerinde çalışacağı maksimum jenerasyon sayısı
- *elitismAmount* : Popülasyondaki bireylerin kaç tanesinin elit birey olarak seçileceğini ifade eder.
- *elderAmount* : Popülasyondaki elit bireylerin kaç tanesinin bir sonraki nesile direk geçeceğini ifade eder.
- *childrenAmount* : Popülasyondaki elit bireylerden kaç tane çocuk üretileceğini ifade eder.
- *randomIndividualAmount* : Popülasyondaki kaç adet rasgele seçilen bireyin bir sonraki nesle aktarılacağını ifade eder.
- *crossoverRate* : 0-1.0 arasındaki olasılıksal değer olarak çaprazlama oranını ifade eder.
- *mutationRate* : 0-1.0 arasındaki olasılıksal değer olarak mutasyon oranını ifade eder.
- *parameterRandomizationTimes* : Bu değişkende belirtilen kadar farklı problem tanımı için program çalıştırılacaktır.
- *problemSolutionTimes* : Her problem tanımı için genetik algoritmanın kaç kez çalıştırıldığını ifade etmektedir.

Hiper-parametre tanımlarının altında ise problemin tanımına global değişken üzerinden erişilmesini görüyoruz.

```java
//Define the multi-metrics array for different parameter sets and solution metrics for the algorithm
ArrayList<ArrayList<Double>> multiMetrics = new ArrayList<ArrayList<Double>>();
ArrayList<ArrayList<Double>> multiResultsAvg = new ArrayList<ArrayList<Double>>();

//Run the genetic algorithm for a predefined number of times with different parameter sets
for(int i = 0; i<parameterRandomizationTimes; i++) {

    ArrayList<Double> resultsAvgFitness = new ArrayList<Double>();

    //Solve the problem for a predefined number of times
    for(int j = 0; j<problemSolutionTimes; j++) {

        ArrayList<Individual> population = OptimizationBuilder.runGA(problemA, populationSize, maxGenerations,
                elitismAmount, elderAmount, randomIndividualAmount,
                childrenAmount, crossoverRate, mutationRate);

        double fitness = Fitness.getAveragePopulationFitness(population);
        resultsAvgFitness.add(fitness);

    }

    //Calculate the metrics for the solutions and save them to the multimetrics array
    double average = MathSupport.average(resultsAvgFitness);
    double bestResult = MathSupport.getBestResult(resultsAvgFitness);
    double worstResult = MathSupport.getWorstResult(resultsAvgFitness);
    double median = MathSupport.median(resultsAvgFitness);
    double stdDev = MathSupport.sd(resultsAvgFitness);

    ArrayList<Double> metrics = new ArrayList<Double>();
    metrics.add(average);
    metrics.add(stdDev);
    metrics.add(bestResult);
    metrics.add(worstResult);
    metrics.add(median);

    multiMetrics.add(metrics);
    multiResultsAvg.add(resultsAvgFitness);
}


//Show metrics for the Genetic Algorithm solution
System.out.println("\n GENETIC ALGORITHM SOLUTION METRICS : \n");
PrintSupport.printMultiMetrics(problemA, multiResultsAvg, multiMetrics);
```

Yukarıda basitçe anlatmak gerekirse, problemlerin çözümleriyle ilgili gerekli çözüm metriklerini tutan ArrayList objeleri tanımlanmış, ardından belirtilen farklı problem tanımları ve çözüm sayıları süresinde genetik algoritmanın belirlenen parametrelerce çalışmasını sağlayan döngü yapıları kullanılmıştır. Programın çalıştırılmasında OptimizationBuilder sınıfındaki kılıf metotlardan yararlanılmıştır. Her tur çözümün ardından gerekli metrikleri ArrayList objelerine kaydedilmiştir.

```java
package opt_builder;

import java.util.ArrayList;

public class OptimizationBuilder {


    /*
     *
     * OPTIMIZATION BUILDER IS A CLASS THAT CONTAINS WRAPPER FUNCTIONS THAT RUNS THE FOLLOWING OPTIMIZATION ALGORITHMS:
     *      -> GENETIC ALGORITHM
     *      -> CONTINUOUS OPTIMIZATION -> DIFFERENTIAL EVOLUTION IS USED
     *              ->> DE/RAND/1
     *              ->> DE/RAND/2
     *              ->> DE/BEST/1
     *              ->> DE/BEST/2
     *              ->> DE/CURRENT_TO_BEST/1
     *
     *
     */


    //This method is used for iteratively pushing the generations in the genetic algorithm further. Defined as a separate function to increase modularity.
    public static ArrayList<Individual> nextGenerationGA(Problem problem, ArrayList<Individual> population,
                                                         double elitismAmount, double elderAmount,
                                                         double randomIndividualAmount, double childrenAmount,
                                                         double crossoverRate, double mutationRate) {

        //evaluate population fitness
        population = Fitness.evaluatePopulationFitness(problem, population);

        //show average population fitness
        Fitness.showAveragePopulationFitness(problem, population);

        //select best individuals/elites from the population
        ArrayList<Individual> elites = Selection.selectElitesByFitness(population, elitismAmount);

        //select elders from elites
        ArrayList<Individual> elders = Selection.selectElders(population, elites, elderAmount);

        //select random individuals from the population
        ArrayList<Individual> randomIvs = Selection.randomlySelectIndividuals(population, randomIndividualAmount);

        //produce children from the elite individuals
        ArrayList<Individual> children = Mating.breedPopulation(problem, elites, childrenAmount, crossoverRate);

        //combine the population
        ArrayList<Individual> combinedPopulation = Population.combinePopulation(elders, children, randomIvs);

        //mutate the children
        children = Mutation.mutatePopulation(problem, combinedPopulation, mutationRate);

        //return the resulting population (children)
        return children;
    }
```

Her jenerasyonda genetik algoritmanın işlemesiyle ilgili adımları ilerleten bir kapsayıcı metot kılıfı olarak dizayn edilen *nextGenerationGA* metotunu yukarıda inceleyebiliriz. Sırasıyla uyumluluk ölçümleme, seçim yapma, yavru üretme ve mutasyon gibi işlevleri sağlayan mutasyonların çağrılması ve jenerasyon ilerlemesi bu sınıfta gerçekleşmektedir. Bu metotu kontrol eden temel fonksiyonu bir sonraki görselde inceleyeceğiz.

```java
//This is the main wrapper function to be able to use for running the genetic algorithm with predefined hyper-parameters
public static ArrayList<Individual> runGA(Problem problem, int populationSize, int maxGenerations,
                            double elitismAmount, double elderAmount,
                            double randomIndividualAmount, double childrenAmount,
                            double crossoverRate, double mutationRate) {

    //generate initial population
    ArrayList<Individual> population = Population.generatePopulation(problem, populationSize);

    //evaluate initial population fitness
    Fitness.evaluatePopulationFitness(problem, population);

    //show initial average population fitness
    Fitness.showAveragePopulationFitness(problem, population);

    //Run the algorithm for maxGenerations times
    System.out.println("GENERATION 0 : \n");

    for(int i = 0; i<maxGenerations; i++) {

        System.out.println("GENERATION :" + i + " \n");

        ArrayList<Individual> nextPopulation = nextGenerationGA(problem, population, elitismAmount, elderAmount,
                                                                randomIndividualAmount, childrenAmount,
                                                                crossoverRate, mutationRate);

        System.out.println("\n_____");

        population = nextPopulation;
    }

    //evaluate final populationn fitness
    population = Fitness.evaluatePopulationFitness(problem, population);

    //return final population
    return population;

}
```

Genetik algoritmayla ilgili tüm parametreleri alarak, temel çözümü gerçekleştiren makro metot runGA'dır. Yukarıda bu metotun öncelikli olarak başlangıç popülasyonunu üretme ve uyumluluk değerlerini ölçümlemeye ek olarak, belirtilen jenerasyon miktarı boyunca neslin ilerletilmesi adına gerekli metotun çağrılması gibi işlevleri gerçekleştirdiğini gözlemleyebiliriz.

(BÖLÜM - 3)

```java
package continuous_optimization;

import java.util.ArrayList;

public class DEPopulation {


    /*
     *
     * THIS CLASS IS USED FOR GENERATING RANDOMIZED POPULATIONS FOR DIFFERENTIAL EVOLUTION ALGORITHM AND PRINTING SPECIFIC DATA ABOUT THEM
     *
     */


    //this method is used for generating an initial population for the differential evolution algorithm
    public static ArrayList<DEIndividual> generateDEPopulation(Problem problem, int populationSize){

        ArrayList<DEIndividual> population = new ArrayList<DEIndividual>();
        for(int i = 0; i<populationSize; i++) {
            DEIndividual individual = new DEIndividual(problem);
            population.add(individual);
        }

        return population;
    }
```

Yukarıda sürekli optimizasyonun gerçekleştirileceği popülasyonun üretilmesi ile ilgili görevi üstlenen fonksiyonu görüyoruz. Bu fonksiyon Differential Optimization kelimelerinin kısaltılmış hali olan DE'den türetilmiş DEPopulation sınıfına ait bir metottur. Belirtilen popülasyon boyutu kadar olacak şekilde DEIndividual (Bireyler)'in üretilmesi görevini üstlenmektedir. DEIndividual sınıfına iki sonraki görselde değineceğiz.

```java
    //this method is used for showing the general data about the population
    public static void showDEPopulationData(ArrayList<DEIndividual> population) {

        System.out.println();
        System.out.println("PRINTING THE INFORMATION OF THE POPULATION ... ");
        System.out.println();
        for(int i = 0; i<population.size(); i++) {
            DEIndividual iv = population.get(i);
            System.out.println("INDIVIDUAL NUMBER: " + i);
            System.out.println("INDIVIDUAL FITNESS SCORE: " + iv.getFitness());
            iv.showChromosome();
            System.out.println("_________________");
        }
    }
```

Bu metot popülasyonun içerisindeki bireylere ait uyumluluk değerlerin görüntülenmesini sağlamaktadır.

```java
package continuous_optimization;

import java.util.ArrayList;

public class DEIndividual implements Comparable<DEIndividual> {

    /*
     *
     * THIS CLASS IS USED FOR DEFINING THE INDIVIDUAL OBJECT, WHICH IS THE MAIN POPULATION ACTOR IN DIFFERENTIAL EVOLUTION ALGORITHM
     *
     */


    private int id;
    private Problem problem;
    private ArrayList<Base> chromosome;
    private Double fitness;

    private static int counter = 0;

    public DEIndividual(Problem problem) {
        counter++;
        this.id = counter;
        this.problem = problem;
        this.chromosome = randomizeChromosome();
        this.fitness = 0.0;
    }
```

DE birey sınıfı yapı olarak genetik algoritma paketine ait birey yapısına benzemek ile birlikte işlevsel farklılığı açısından sürekli optimizasyon modülünde olması daha uygun görülmüştür. Aynı şekilde baz istasyonu lokasyonlarını içeren bir kromozoma ve bir uygunluk değerine sahiptir.

```java
//this method is used for randomizing the chromosome (base coordinates) of the individual on creation
public ArrayList<Base> randomizeChromosome(){

    Random r = new Random();
    double x;
    double y;
    ArrayList<Base> bases = new ArrayList<Base>();
    if(problem.getCity().getBaseLocationAmount() == 0) {

        for(int i = 0; i<problem.getCity().getBaseAmount(); i++) {

            //create random locations in a continous space if the city has no predefined base locations
            x = problem.getCity().getWidth() * r.nextDouble();
            y = problem.getCity().getHeight() * r.nextDouble();
            Base base = new Base(x, y, problem.getCoverRadius());
            bases.add(base);
        }

        return bases;
    }
    else {

        for(BaseLocation location  : problem.getBaseLocations()) {
            location.setOccupied(false);
        }

        int choice;
        for(int i = 0; i<problem.getCity().getBaseAmount(); i++) {

            choice = r.nextInt(problem.getBaseLocations().size());
            if(problem.getBaseLocations().get(choice).isOccupied() == true) {
                i = i -1;
                continue;
            }
            else {

                //select from predefined base locations if they exist
                x = problem.getBaseLocations().get(choice).getX();
                y = problem.getBaseLocations().get(choice).getY();
                Base base = new Base(x, y, problem.getCoverRadius());
                bases.add(base);
                problem.getBaseLocations().get(choice).setOccupied(true);
            }
        }

        return bases;
    }
}
```

DE bireylerinin randomize kromozom üretimi yukarıdaki şekilde görüldüğü üzere gerçekleşmektedir. Problem tanımında belirtildiği üzere mevcut olmayan baz lokasyonları kaynaklı olarak, sürekli bir 2 boyutlu düzlem üzerinde rasgele x(yatay) ve y(dikey) noktaları seçilmesi yoluyla çalışmaktadır.

```java
//this method is used for printing the chromosome information (base coordintes) of the individual
public void showChromosome() {
    System.out.println();
    System.out.println("\n INDIVIDUAL BASE COORDINATES : ");
    for(int i = 0; i<chromosome.size(); i++) {
        Base base = chromosome.get(i);
        System.out.print("Base "+i+" ("+
                         String.format("%.2f", base.getX())+
                         " / "+
                         String.format("%.2f", base.getY())+
                         ") , ");
    }
    System.out.println();
}
```

Yukarıdaki metot ise DE bireylerinin kromozomlarını, yani sahip oldukları baz istasyonu noktalarının koordinatlarını görüntüleyebilmek için tasarlanmıştır.

```java
package continuous_optimization;

import java.util.ArrayList;

public class DEFitness {


    /*
     *
     * THIS CLASS IS USED FOR EVALUATING THE FITNESS LEVEL OF THE INDIVIDUALS FOR DIFFERENTIAL EVOLUTION ALGORITHM
     *
     */



    //this method is used for evaluating the fitness score of a single individual
    public static double evaluateDEIndividualFitness(Problem problem, DEIndividual iv) {

        ArrayList<Customer> customers = problem.getCustomers();

        double fitness = 0;
        for(int j = 0; j<customers.size(); j++) {

            Customer customer = customers.get(j);
            for(int k = 0; k<iv.getChromosome().size(); k++) {

                Base base = iv.getChromosome().get(k);

                //take distance and coverage status of each customer in the city.
                double deltaXPow = Math.pow(customer.getX() - base.getX(), 2);
                double deltaYPow = Math.pow(customer.getY() - base.getY(), 2);
                double sqrtSum = Math.sqrt((deltaXPow + deltaYPow));

                //evaluate fitnesss score
                if(sqrtSum <= problem.getCoverRadius()){
                    fitness = fitness + 1;
                    break;
                }else {
                    continue;
                }
            }
        }

        iv.setFitness(fitness);

        return fitness;
    }
```

Bu metot, DE algoritmasına ait tek bir bireyin uyumluluk değerinin ölçümlenmesini sağlamaktadır. Genetik algoritmaya benzer şekilde uzaklık dahilinde yer alan müşterilerin kapsanıp kapsanmadığının ölçümlenmesi ilkesine dayalı olarak bir uyumluluk değeri belirlenmektedir.

```java
  //this method is used for evaluating the fitness score of the entire differential evoliton population arraylist
  public static ArrayList<DEIndividual> evaluateDEPopulationFitness(Problem problem, ArrayList<DEIndividual> population){

      for(int i = 0; i<population.size(); i++) {

          DEIndividual iv = population.get(i);

          //evaluate the fitness of each individual
          double fitness = evaluateDEIndividualFitness(problem, iv);


          //set fitness score of each individual
          iv.setFitness(fitness);
      }

      return population;
  }
```

DE Birey uyumluluk ölçümleme metodunun kılıflayıcısı olan metottur. Bu sayede popülasyona ait her bireyin uyumluluk değerinin ölçümlenmesi sağlanmaktadır.

```java
//this method is used for showing the population scores of the each individual in the differential evolution population
public static void showDEPopulationFitnessScores(Problem problem, ArrayList<DEIndividual> population) {

    System.out.println();
    System.out.println("SHOWING POPULATION FITNESS SCORES ...");
    System.out.println();
    for(int i = 0; i<population.size(); i++) {
        double prc = (population.get(i).getFitness()/problem.getCity().getCustomerAmount())*100;
        System.out.println("INDIVIDUAL "+i+" FITNESS SCORE : " + population.get(i).getFitness() + " / "+ problem.getCity().getCustomerAmount() + " ( %" + prc + " )
    }
    System.out.println();
}


//this method is used for getting the average population fitenss score in the given population
public static double getAverageDEPopulationFitness(ArrayList<DEIndividual> population) {

    double summation = 0;
    for(int i = 0; i<population.size(); i++) {

        summation = summation + population.get(i).getFitness();
    }

    double average = (double) (summation/population.size());

    return average;
}


//this method is used for printing the average population fitness score in the given population
public static void showAverageDEPopulationFitness(Problem problem, ArrayList<DEIndividual> population) {

    double summation = 0;
    for(int i = 0; i<population.size(); i++) {

        summation = summation + population.get(i).getFitness();
    }

    double average = summation/population.size();

    double prc = (average/problem.getCity().getCustomerAmount())*100;
    System.out.println("AVERAGE FITNESS OF POPULATION : "+average+" / "+problem.getCity().getCustomerAmount() + " ( %" + prc + " )");
}
```

Yukarıdaki metotlar popülasyona ait bireylerin uyumluluk değerlerinin tek tek listelenmesi ve ekranda görüntülenmesi, ortalama popülasyon uygunluk değerine erişilmesi veya ekranda görüntülenmesini sağlayan metotlardır.

```java
package continuous_optimization;

import java.util.ArrayList;

public class DECrossover {

    /*
     *
     * THIS CLASS IS USED FOR APPLYING MUTATION TO THE DIFFERENTIAL EVOLUTION POPULATION BASED ON PARAMETERS
     *
     */


    //this method is used for mutating the differential evolution population with the specified method : DE/rand/1
    public static ArrayList<DEIndividual> mutateDEPopulation_DErand1(Problem problem, ArrayList<DEIndividual> population, double F, double crossoverRate){

        Random r = new Random();

        ArrayList<DEIndividual> mutatedPopulation = new ArrayList<DEIndividual>();
        for (int i = 0; i<population.size(); i++) {

            DEIndividual deiv = population.get(i);
            int r1Idx = -1;
            int r2Idx = -1;
            int r3Idx = -1;

            //specify indexes for the selected population individuals
            int idx = r.nextInt(population.size());
            while(idx == i)
                idx = r.nextInt(population.size());
            r1Idx = idx;
            while(idx == i || idx == r1Idx)
                idx = r.nextInt(population.size());
            r2Idx = idx;
            while(idx == i || idx == r1Idx || idx == r2Idx)
                idx = r.nextInt(population.size());
            r3Idx = idx;

            DEIndividual r1 = population.get(r1Idx);
            DEIndividual r2 = population.get(r2Idx);
            DEIndividual r3 = population.get(r3Idx);

            //mutate the individuals
            DEIndividual newIndividual = DECrossover.mutateDEIndividual_DErand1(problem, deiv, r1, r2, r3, F, crossoverRate);
            mutatedPopulation.add(newIndividual);
        }

        //return mutated individual
        return mutatedPopulation;
    }
```

Yukarıda DECrossover sınıfına ait bir ekran görüntüsü görmekteyiz. Bu sınıfın içerisinde çeşitli farklı mutasyon yöntemleri kullanılarak sürekli optimizasyon sağlamayı amaçlayan statik metotlar yer almaktadır. Yukarıda **DE/rand/1** mutasyon türünü popülasyon üzerine uygulayan metot yer almaktadır. Bu metot, içeriğinde ilgili birey mutasyon fonksiyonunu çağırmaktadır

```java
    //this method is used for mutating the differential evolution population with the specified method : DE/rand/2
    public static ArrayList<DEIndividual> mutateDEPopulation_DErand2(Problem problem, ArrayList<DEIndividual> population, double F, double crossoverRate){

        Random r = new Random();

        ArrayList<DEIndividual> mutatedPopulation = new ArrayList<DEIndividual>();
        for (int i = 0; i<population.size(); i++) {

            DEIndividual deiv = population.get(i);
            int r1Idx = -1;
            int r2Idx = -1;
            int r3Idx = -1;
            int r4Idx = -1;
            int r5Idx = -1;

            //specify indexes for the selected population individuals
            int idx = r.nextInt(population.size());
            while(idx == i)
                idx = r.nextInt(population.size());
            r1Idx = idx;
            while(idx == i || idx == r1Idx)
                idx = r.nextInt(population.size());
            r2Idx = idx;
            while(idx == i || idx == r1Idx || idx == r2Idx)
                idx = r.nextInt(population.size());
            r3Idx = idx;
            while(idx == i || idx == r1Idx || idx == r2Idx || idx == r3Idx)
                idx = r.nextInt(population.size());
            r4Idx = idx;
            while(idx == i || idx == r1Idx || idx == r2Idx || idx == r3Idx || idx == r4Idx  )
                idx = r.nextInt(population.size());
            r5Idx = idx;

            DEIndividual r1 = population.get(r1Idx);
            DEIndividual r2 = population.get(r2Idx);
            DEIndividual r3 = population.get(r3Idx);
            DEIndividual r4 = population.get(r4Idx);
            DEIndividual r5 = population.get(r5Idx);

            //mutate the individuals
            DEIndividual newIndividual = DECrossover.mutateDEIndividual_DErand2(problem, deiv, r1, r2, r3, r4, r5, F, crossoverRate);
            mutatedPopulation.add(newIndividual);
        }

        //return mutated population
        return mutatedPopulation;
    }
```

DE/rand/2 “popülasyon” mutasyon fonksiyonu

```java
//this method is used for mutating the differential evolution population with the specified method : DE/best/1
public static ArrayList<DEIndividual> mutateDEPopulation_DEbest1(Problem problem, ArrayList<DEIndividual> population, double F, double crossoverRate){

    Random r = new Random();

    DEIndividual bestDeiv = getBestDEIndividual(population);
    int bestDeivIdx = -1;
    for (int i = 0; i<population.size(); i++) {
        if( population.get(i) == bestDeiv)
            bestDeivIdx = i;
    }

    ArrayList<DEIndividual> mutatedPopulation = new ArrayList<DEIndividual>();
    for (int i = 0; i<population.size(); i++) {

        DEIndividual deiv = population.get(i);
        int r1Idx = -1;
        int r2Idx = -1;

        //specify different indexes for the population individuals
        int idx = r.nextInt(population.size());
        while(idx == i || idx == bestDeivIdx )
            idx = r.nextInt(population.size());
        r1Idx = idx;
        while(idx == i || idx == bestDeivIdx || idx == r1Idx )
            idx = r.nextInt(population.size());
        r2Idx = idx;

        DEIndividual r1 = population.get(r1Idx);
        DEIndividual r2 = population.get(r2Idx);

        //mutate each individual
        DEIndividual newIndividual = DECrossover.mutateDEIndividual_DEbest1(problem, deiv, bestDeiv, r1, r2, F, crossoverRate);
        mutatedPopulation.add(newIndividual);
    }

    //returnmutated population
    return mutatedPopulation;
}
```

DE/best/1 “popülasyon” mutasyon fonksiyonu

```java
//this method is used for mutating the differential evolution popoulation with the specified method : DE/best/2
public static ArrayList<DEIndividual> mutateDEPopulation_DEbest2(Problem problem, ArrayList<DEIndividual> population, double F, double crossoverRate){


    Random r = new Random();

    DEIndividual bestDeiv = getBestDEIndividual(population);
    int bestDeivIdx = -1;
    for (int i = 0; i<population.size(); i++) {
        if( population.get(i) == bestDeiv)
            bestDeivIdx = i;
    }

    ArrayList<DEIndividual> mutatedPopulation = new ArrayList<DEIndividual>();
    for (int i = 0; i<population.size(); i++) {

        DEIndividual deiv = population.get(i);
        int r1Idx = -1;
        int r2Idx = -1;
        int r3Idx = -1;
        int r4Idx = -1;


        //specify different indexes for population individuals
        int idx = r.nextInt(population.size());
        while(idx == i || idx == bestDeivIdx )
            idx = r.nextInt(population.size());
        r1Idx = idx;
        while(idx == i || idx == bestDeivIdx || idx == r1Idx )
            idx = r.nextInt(population.size());
        r2Idx = idx;
        while(idx == i || idx == bestDeivIdx || idx == r1Idx || idx == r2Idx  )
            idx = r.nextInt(population.size());
        r3Idx = idx;
        while(idx == i || idx == bestDeivIdx || idx == r1Idx || idx == r2Idx || idx == r3Idx)
            idx = r.nextInt(population.size());
        r4Idx = idx;

        DEIndividual r1 = population.get(r1Idx);
        DEIndividual r2 = population.get(r2Idx);
        DEIndividual r3 = population.get(r3Idx);
        DEIndividual r4 = population.get(r4Idx);

        //mutate each individual
        DEIndividual newIndividual = DECrossover.mutateDEIndividual_DEbest2(problem, deiv, bestDeiv, r1, r2, r3, r4, F, crossoverRate);
        mutatedPopulation.add(newIndividual);
    }

    //return mutated population
    return mutatedPopulation;
}
```

DE/best/2 “popülasyon” mutasyon fonksiyonu

```java
//this method is used for mutating the differential evolution population with the specified method : DE/current_to_best/1
public static ArrayList<DEIndividual> mutateDEPopulation_DEcurToBest1(Problem problem, ArrayList<DEIndividual> population, double F, double crossoverRate){


    Random r = new Random();

    DEIndividual bestDeiv = getBestDEIndividual(population);
    int bestDeivIdx = -1;
    for (int i = 0; i<population.size(); i++) {
        if( population.get(i) == bestDeiv)
            bestDeivIdx = i;
    }

    ArrayList<DEIndividual> mutatedPopulation = new ArrayList<DEIndividual>();
    for (int i = 0; i<population.size(); i++) {

        DEIndividual deiv = population.get(i);
        int r1Idx = -1;
        int r2Idx = -1;

        //specify different indexes for population individuals
        int idx = r.nextInt(population.size());
        while(idx == i || idx == bestDeivIdx )
            idx = r.nextInt(population.size());
        r1Idx = idx;
        while(idx == i || idx == bestDeivIdx || idx == r1Idx )
            idx = r.nextInt(population.size());
        r2Idx = idx;

        DEIndividual r1 = population.get(r1Idx);
        DEIndividual r2 = population.get(r2Idx);

        //mutate each individual
        DEIndividual newIndividual = DECrossover.mutateDEIndividual_DEbest1(problem, deiv, bestDeiv, r1, r2, F, crossoverRate);
        mutatedPopulation.add(newIndividual);
    }

    //return mutated population
    return mutatedPopulation;
}
```

DE/current_to_best/1 “popülasyon” mutasyon fonksiyonu

Bu fonksiyonlar ortak olarak mutasyon ve deneme vektörlerini kullanmaktadır. Bu kaynakların birbirinden farklı olmasına dikkat edilmekte ve sadece deneme vektörünün uygunluk avantajı sağladığı koşulda popülasyona iletilmesi sağlanmaktadır. Birbirlerinden farkları ise mutant vektörün oluşturulması sırasında dikkate alınan birey sayısı, bireyin niteliği (örneğin : en iyi birey, şu anki birey vb.) rol oynamaktadır. Sonraki görsellerde, tek bir bireyin mutasyonunu üreten fonksiyonları görüntüleyeceğiz.

```java
//this method is used for mutating a single individual inside the population with DE/rand/1 method
public static DEIndividual mutateDEIndividual_DErand1(Problem problem,
                                                       DEIndividual deiv,
                                                       DEIndividual r1,
                                                       DEIndividual r2,
                                                       DEIndividual r3,
                                                       double F,
                                                       double crossoverRate) {


    DEIndividual mutatedIndividual = new DEIndividual(problem);

    ArrayList<Base> mutatedChromosome = new ArrayList<Base>();

    Random r = new Random();
    crossoverRate = crossoverRate * 100;

    //define the mutaton index
    int mutationIndex = r.nextInt(deiv.getChromosome().size());


    for (int i = 0; i<deiv.getChromosome().size(); i++) {

        int coDie = r.nextInt(100);

        //create the mutant vectors
        double mutantX = r1.getChromosome().get(i).getX() + (F * (r2.getChromosome().get(i).getX() - r3.getChromosome().get(i).getX()));
        double mutantY = r1.getChromosome().get(i).getY() + (F * (r2.getChromosome().get(i).getY() - r3.getChromosome().get(i).getY()));

        double trialX;
        double trialY;

        //create the trial vector
        if (coDie < crossoverRate || i == mutationIndex) {

            trialX = mutantX;
            trialY = mutantY;

        }
        else {

            trialX = deiv.getChromosome().get(i).getX();
            trialY = deiv.getChromosome().get(i).getY();

        }

        //create the mutated base and add to chromosome
        Base mutatedBase = new Base(trialX, trialY, problem.getCoverRadius());

        mutatedChromosome.add(mutatedBase);
    }

    mutatedIndividual.setChromosome(mutatedChromosome);

    double newFitness = DEFitness.evaluateDEIndividualFitness(problem, mutatedIndividual);
    double oldFitness = deiv.getFitness();

    //if the new fitness is better, save it
    if(newFitness <= oldFitness) {
        mutatedIndividual = deiv;
    }
    else {
        mutatedIndividual.setChromosome(mutatedChromosome);
    }

    //return mutated individual
    return mutatedIndividual;

}
```

DE/rand/1 “birey” mutasyon fonksiyonu

```
//this method is used for mutating a single individual in the population with DE/rand/2 method
public static DEIndividual mutateDEIndividual_DErand2(Problem problem,
                                                       DEIndividual deiv,
                                                       DEIndividual r1,
                                                       DEIndividual r2,
                                                       DEIndividual r3,
                                                       DEIndividual r4,
                                                       DEIndividual r5,
                                                       double F,
                                                       double crossoverRate) {

    DEIndividual mutatedIndividual = new DEIndividual(problem);

    ArrayList<Base> mutatedChromosome = new ArrayList<Base>();

    Random r = new Random();
    crossoverRate = crossoverRate * 100;

    int mutationIndex = r.nextInt(deiv.getChromosome().size());


    for (int i = 0; i<deiv.getChromosome().size(); i++) {

        int coDie = r.nextInt(100);

        //define the mutant vectors
        double mutantX = r1.getChromosome().get(i).getX() + (F * (r2.getChromosome().get(i).getX() - r3.getChromosome().get(i).getX()) +
                        F * (r4.getChromosome().get(i).getX() - r5.getChromosome().get(i).getX()));


        double mutantY = r1.getChromosome().get(i).getY() + (F * (r2.getChromosome().get(i).getY() - r3.getChromosome().get(i).getY()) +
                F * (r4.getChromosome().get(i).getY() - r5.getChromosome().get(i).getY()));

        double trialX;
        double trialY;


        //create the trial vectors
        if (coDie < crossoverRate || i == mutationIndex) {

            trialX = mutantX;
            trialY = mutantY;

        }
        else {

            trialX = deiv.getChromosome().get(i).getX();
            trialY = deiv.getChromosome().get(i).getY();

        }

        //create the mutated base and add it to the chromosome
        Base mutatedBase = new Base(trialX, trialY, problem.getCoverRadius());

        mutatedChromosome.add(mutatedBase);
    }

    mutatedIndividual.setChromosome(mutatedChromosome);

    double newFitness = DEFitness.evaluateDEIndividualFitness(problem, mutatedIndividual);
    double oldFitness = deiv.getFitness();

    //if the new fitness is better, save it
    if(newFitness <= oldFitness) {
        mutatedIndividual = deiv;
    }
    else {
        mutatedIndividual.setChromosome(mutatedChromosome);
    }

    //return the mutated individual
    return mutatedIndividual;

}
```

DE/rand/2 “birey” mutasyon fonksiyonu

```java
//this method is used for mutating a single individual with DE/best/1 method
public static DEIndividual mutateDEIndividual_DEbest1(Problem problem,
                                                       DEIndividual deiv,
                                                       DEIndividual bestDeiv,
                                                       DEIndividual r1,
                                                       DEIndividual r2,
                                                       double F,
                                                       double crossoverRate) {


    DEIndividual mutatedIndividual = new DEIndividual(problem);

    ArrayList<Base> mutatedChromosome = new ArrayList<Base>();

    Random r = new Random();
    crossoverRate = crossoverRate * 100;

    int mutationIndex = r.nextInt(deiv.getChromosome().size());


    for (int i = 0; i<deiv.getChromosome().size(); i++) {

        int coDie = r.nextInt(100);

        //create the mutant vectors
        double mutantX = bestDeiv.getChromosome().get(i).getX() + (F * (r1.getChromosome().get(i).getX() - r2.getChromosome().get(i).getX()));
        double mutantY = bestDeiv.getChromosome().get(i).getY() + (F * (r1.getChromosome().get(i).getY() - r2.getChromosome().get(i).getY()));

        double trialX;
        double trialY;


        //create the trial vectors
        if (coDie < crossoverRate || i == mutationIndex) {

            trialX = mutantX;
            trialY = mutantY;

        }
        else {

            trialX = deiv.getChromosome().get(i).getX();
            trialY = deiv.getChromosome().get(i).getY();

        }

        //create the mutated base and add it to the chromosome
        Base mutatedBase = new Base(trialX, trialY, problem.getCoverRadius());

        mutatedChromosome.add(mutatedBase);
    }

    mutatedIndividual.setChromosome(mutatedChromosome);

    double newFitness = DEFitness.evaluateDEIndividualFitness(problem, mutatedIndividual);
    double oldFitness = deiv.getFitness();

    //if the new fitness is better, save it
    if(newFitness <= oldFitness) {
        mutatedIndividual = deiv;
    }
    else {
        mutatedIndividual.setChromosome(mutatedChromosome);
    }

    //return mutated individual
    return mutatedIndividual;
}
```

DE/best/1 “birey” mutasyon fonksiyonu

```java
//this method is used for mutating a single individual in the population with DE/best/2 method
public static DEIndividual mutateDEIndividual_DEbest2(Problem problem,
                                                       DEIndividual deiv,
                                                       DEIndividual bestDeiv,
                                                       DEIndividual r1,
                                                       DEIndividual r2,
                                                       DEIndividual r3,
                                                       DEIndividual r4,
                                                       double F,
                                                       double crossoverRate) {


    DEIndividual mutatedIndividual = new DEIndividual(problem);

    ArrayList<Base> mutatedChromosome = new ArrayList<Base>();

    Random r = new Random();
    crossoverRate = crossoverRate * 100;

    int mutationIndex = r.nextInt(deiv.getChromosome().size());


    for (int i = 0; i<deiv.getChromosome().size(); i++) {

        int coDie = r.nextInt(100);

        //create mutant vectors
        double mutantX = bestDeiv.getChromosome().get(i).getX() + (F * (r1.getChromosome().get(i).getX() - r2.getChromosome().get(i).getX()) +
                (F * (r3.getChromosome().get(i).getX() - r4.getChromosome().get(i).getX())));
        double mutantY = bestDeiv.getChromosome().get(i).getY() + (F * (r1.getChromosome().get(i).getY() - r2.getChromosome().get(i).getY()) +
                (F * (r3.getChromosome().get(i).getY() - r4.getChromosome().get(i).getY())));

        double trialX;
        double trialY;


        //create trial vectors
        if (coDie < crossoverRate || i == mutationIndex) {

            trialX = mutantX;
            trialY = mutantY;

        }
        else {

            trialX = deiv.getChromosome().get(i).getX();
            trialY = deiv.getChromosome().get(i).getY();

        }

        //create the mutated base and save it to the chromosome
        Base mutatedBase = new Base(trialX, trialY, problem.getCoverRadius());

        mutatedChromosome.add(mutatedBase);
    }

    mutatedIndividual.setChromosome(mutatedChromosome);

    double newFitness = DEFitness.evaluateDEIndividualFitness(problem, mutatedIndividual);
    double oldFitness = deiv.getFitness();

    //if the new fitness is better, save it
    if(newFitness <= oldFitness) {
        mutatedIndividual = deiv;
    }
    else {
        mutatedIndividual.setChromosome(mutatedChromosome);
    }

    //return the mutated individual
    return mutatedIndividual;
}
```

DE/best/2 “birey” mutasyon fonksiyonu

```java
//this method is used for applying mutation to a single individual with DE/current_to_best/1 method
public static DEIndividual mutateDEIndividual_DEcurToBest1(Problem problem,
                                                            DEIndividual deiv,
                                                            DEIndividual bestDeiv,
                                                            DEIndividual r1,
                                                            DEIndividual r2,
                                                            double F,
                                                            double crossoverRate) {


    DEIndividual mutatedIndividual = new DEIndividual(problem);

    ArrayList<Base> mutatedChromosome = new ArrayList<Base>();

    Random r = new Random();
    crossoverRate = crossoverRate * 100;

    int mutationIndex = r.nextInt(deiv.getChromosome().size());


    for (int i = 0; i<deiv.getChromosome().size(); i++) {

        int coDie = r.nextInt(100);

        //create the mutant vectors
        double mutantX = deiv.getChromosome().get(i).getX() + (F * (bestDeiv.getChromosome().get(i).getX() - deiv.getChromosome().get(i).getX()) +
                (F * (r1.getChromosome().get(i).getX() - r2.getChromosome().get(i).getX())));
        double mutantY = deiv.getChromosome().get(i).getY() + (F * (bestDeiv.getChromosome().get(i).getY() - deiv.getChromosome().get(i).getY()) +
                (F * (r1.getChromosome().get(i).getY() - r2.getChromosome().get(i).getY())));

        double trialX;
        double trialY;


        //create the trial vectors
        if (coDie < crossoverRate || i == mutationIndex) {

            trialX = mutantX;
            trialY = mutantY;

        }
        else {

            trialX = deiv.getChromosome().get(i).getX();
            trialY = deiv.getChromosome().get(i).getY();

        }

        //create the mutated base and save it to the chromosome
        Base mutatedBase = new Base(trialX, trialY, problem.getCoverRadius());

        mutatedChromosome.add(mutatedBase);
    }

    mutatedIndividual.setChromosome(mutatedChromosome);

    double newFitness = DEFitness.evaluateDEIndividualFitness(problem, mutatedIndividual);
    double oldFitness = deiv.getFitness();

    //if the new fitness is better, save it
    if(newFitness <= oldFitness) {
        mutatedIndividual = deiv;
    }
    else {
        mutatedIndividual.setChromosome(mutatedChromosome);
    }

    //return mutated individual
    return mutatedIndividual;

}
```

DE/current_to_best/1 “birey” mutasyon fonksiyonu

```java
//this method is used for achieving the individual with the best fitness level in a population
public static DEIndividual getBestDEIndividual(ArrayList<DEIndividual> population) {

    DEIndividual maxDeiv = population.get(0);
    double max = 0;
    for(int i = 0; i<population.size(); i++) {

        //find the individual with the maximum fitness level
        if(population.get(i).getFitness() > max) {
            max = population.get(i).getFitness();
            maxDeiv = population.get(i);
        }
    }

    //return individual with the maximum fitness score
    return maxDeiv;
}
```

Bu metot, popülasyon içerisindeki en iyi bireye erişilmesini sağlayan yardımcı metottur. En yüksek uyumluluk değerini içeren bireyi geri döndürür.

```java
package tests;

import java.util.ArrayList;

public class COGeneralTest {

    public static void main(String[] args) {

        /*
         *  THIS CLASS IS USED FOR TESTING THE USE OF A CONTINUOUS OPTIMIZATION ALGORITHM FOR OPTIMIZING
         *  THE PROBLEM THAT WAS MENTIONED IN "HOMEWORK PART 3 : CITY B"
         *
         *  USED METHOD : DIFFERENTIAL EVOLUTION
         *
         *  I HAVE USED 5 DIFFERENT MUTATION METHODS :
         *  1) DE/RAND/1
         *  2) DE/RAND/2
         *  3) DE/BEST/1
         *  4) DE/BEST/2
         *  5) DE/CURRENT_TO_BEST/1
         *
         *  BY USING THIS PAGE, YOU CAN TEST ALL OF THESE DIFFERENT MUTATION METHODS ON THE SAME PROBLEM.
         *
         */


        //Continuous Optimization Hyper-parameters
        int populationSize = 100;
        int maxGenerations = 2000;
        double F = 0.1;
        double crossoverRate = 0.9;

        //Multi-Solution Hyper-parameters
        double parameterRandomizationTimes = 4;
        double problemSolutionTimes = 25;

        //Achieve the problem from global variables class
        Problem problemB = GlobalDEVariables.problemB;
        problemB.showCustomerCoordinates();

        //Define the wrapper arrays that will hold the metrics from multiple calculations with multiple parameter sets
        ArrayList<ArrayList<Double>> multiMetrics1 = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiMetrics2 = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiMetrics3 = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiMetrics4 = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiMetrics5 = new ArrayList<ArrayList<Double>>();

        ArrayList<ArrayList<Double>> multiResultsAvg1 = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiResultsAvg2 = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiResultsAvg3 = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiResultsAvg4 = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiResultsAvg5 = new ArrayList<ArrayList<Double>>();
```

Yukarıda *COGeneralTest* sınıfını görmekteyiz. Bu sınıf, sürekli optimizasyon, spesifik olarak da Diferansiyel Gelişim algoritmasını çeşitli mutasyon fonksiyonlarıyla test edeceğimiz sınıftır. Yukarıda öncelikle hiper parametre değerlerinin tanımlandığını görüyoruz. Bunları kısaca açıklamak gerekirse;

- *populationSize* : Popülasyondaki birey sayısı
- *maxGenerations* : Bir diferansiyel gelişim algoritmasının uygulanacağı jenerasyon sayısı
- *F* : harmanlanma oranı
- *crossoverRate* : Çaprazlama/Mutasyon oranı
- *parameterRandomizationTimes* : Problemin kaç farklı problem tanımı için çözüleceğini ifade eden parametredir.
- *problemSolutionTimes* : Her problem tanımı için diferansiyel gelişim algoritmasının kaç kez çalışacağını ifade eden değerdir.

Görselin devamında probleme global problem tanımı üzerinden erişildiğini görüyoruz. Ardından ise, çözüm metriklerinin farklı mutasyon fonksiyonları için ayrı ayrı tutulması için gerekli listeler tanımlanmıştır.

```
//Run the continuous optimization algorithm
for(int i = 0; i<parameterRandomizationTimes; i++) {

    //For each parameter set, define 5 array lists that will hold the result metrics of 5 different differential evolution mutation methods.
    ArrayList<Double> resultsAvgFitness1 = new ArrayList<Double>();
    ArrayList<Double> resultsAvgFitness2 = new ArrayList<Double>();
    ArrayList<Double> resultsAvgFitness3 = new ArrayList<Double>();
    ArrayList<Double> resultsAvgFitness4 = new ArrayList<Double>();
    ArrayList<Double> resultsAvgFitness5 = new ArrayList<Double>();

    //Solve the problem for each of the parameter set , for defined number of times.
    for(int j = 0; j<problemSolutionTimes; j++) {

        //Run continuous optimization algorithm to run for predefined maximum epochs (for each of the differential evolution mutation methods)
        ArrayList<DEIndividual> population1 = OptimizationBuilder.runContinuousOptimization_DErand1(problemB, populationSize, maxGenerations, F, crossoverRate);
        ArrayList<DEIndividual> population2 = OptimizationBuilder.runContinuousOptimization_DErand2(problemB, populationSize, maxGenerations, F, crossoverRate);
        ArrayList<DEIndividual> population3 = OptimizationBuilder.runContinuousOptimization_DEbest1(problemB, populationSize, maxGenerations, F, crossoverRate);
        ArrayList<DEIndividual> population4 = OptimizationBuilder.runContinuousOptimization_DEbest2(problemB, populationSize, maxGenerations, F, crossoverRate);
        ArrayList<DEIndividual> population5 = OptimizationBuilder.runContinuousOptimization_DEcurToBest1(problemB, populationSize, maxGenerations, F, crossoverRate);

        //Calculate the fitness levels for each of the mutation methods after maximum epochs reached.
        double fitness1 = DEFitness.getAverageDEPopulationFitness(population1);
        double fitness2 = DEFitness.getAverageDEPopulationFitness(population2);
        double fitness3 = DEFitness.getAverageDEPopulationFitness(population3);
        double fitness4 = DEFitness.getAverageDEPopulationFitness(population4);
        double fitness5 = DEFitness.getAverageDEPopulationFitness(population5);

        //Save the fitness scores to the different arraylists
        resultsAvgFitness1.add(fitness1);
        resultsAvgFitness2.add(fitness2);
        resultsAvgFitness3.add(fitness3);
        resultsAvgFitness4.add(fitness4);
        resultsAvgFitness5.add(fitness5);

    }
```

Belirtilen farklı problem tanımları için, belirtilen çözüm sayıları dikkate alınarak döngü içerisinde 5 farklı mutasyon fonksiyonları kullanılarak problemin çözümlenmesi sağlanır. Kullanılan mutasyon fonksiyonları;

- **DE/rand/1**
- **DE/rand/2**
- **DE/best/1**
- **DE/best/2**
- **DE/current_to_best/1**

Çözümlerle ilgili gerekli metrikler kaydedilir.

```
    //Calculate the metrics for the first mutation method : DE/CURRENT_TO_BEST/1 and save them to the corresponding multi-metrics array
    average = MathSupport.average(resultsAvgFitness5);
    bestResult = MathSupport.getBestResult(resultsAvgFitness5);
    worstResult = MathSupport.getWorstResult(resultsAvgFitness5);
    median = MathSupport.median(resultsAvgFitness5);
    stdDev = MathSupport.sd(resultsAvgFitness5);

    ArrayList<Double> metrics5 = new ArrayList<Double>();
    metrics5.add(average);
    metrics5.add(stdDev);
    metrics5.add(bestResult);
    metrics5.add(worstResult);
    metrics5.add(median);

    multiMetrics5.add(metrics5);
    multiResultsAvg5.add(resultsAvgFitness5);


}

//Show metrics for DE/RAND/1
System.out.println("\n MUTATION METHOD -> DE/RAND/1 -> METRICS \n ");
PrintSupport.printMultiMetrics(problemB, multiResultsAvg1, multiMetrics1);

//Show metrics for DE/RAND/2
System.out.println("\n MUTATION METHOD -> DE/RAND/2 -> METRICS \n ");
PrintSupport.printMultiMetrics(problemB, multiResultsAvg2, multiMetrics2);

//Show metrics for DE/BEST/1
System.out.println("\n MUTATION METHOD -> DE/BEST/1 -> METRICS \n ");
PrintSupport.printMultiMetrics(problemB, multiResultsAvg3, multiMetrics3);

//Show metrics for DE/BEST/2
System.out.println("\n MUTATION METHOD -> DE/BEST/2 -> METRICS \n ");
PrintSupport.printMultiMetrics(problemB, multiResultsAvg4, multiMetrics4);

//Show metrics for DE/CURRENT_TO_BEST/1
System.out.println("\n MUTATION METHOD -> DE/CURRENT_TO_BEST/1 -> METRICS \n ");
PrintSupport.printMultiMetrics(problemB, multiResultsAvg5, multiMetrics5);
```

Her çözüm değeri için gerekli istatistikler hesaplanır ve ekranda görüntülenerek kullanıcıya bildirilir.

```java
//This is the main wrapper function to be able to use for running the differential evolution (DE/rand/1) with predefined hyper-parameters
public static ArrayList<DEIndividual> runContinuousOptimization_DErand1(Problem problem, int populationSize, int maxGenerations,
                                                                         double F, double crossoverRate) {

    //Generate initial population
    ArrayList<DEIndividual> population = DEPopulation.generateDEPopulation(problem, populationSize);

    //evaluate initial population fitness
    DEFitness.evaluateDEPopulationFitness(problem, population);

    //show average initial population fitness
    DEFitness.showAverageDEPopulationFitness(problem, population);

    //run the algorithm for maxGenerations times
    System.out.println("GENERATION 0 : \n");

    for(int i = 0; i<maxGenerations; i++) {

        System.out.println("GENERATION :" + i + " \n");

        ArrayList<DEIndividual> nextPopulation = nextGenerationContinuousOptimization_DErand1(problem, population, F, crossoverRate);

        System.out.println("\n_____");

        population = nextPopulation;
    }

    //Evaluate final population fitness
    population = DEFitness.evaluateDEPopulationFitness(problem, population);

    //return final population
    return population;
}
```

Sürekli optimizasyon algoritmasının çalışması için gerekli jenerasyon ilerletimi faaliyetini üstlenen ve *OptimizationBuilder* sınıfına dahil olan metot yukarıdaki gibidir. Başlangıç jenerasyonunun üretilmesi, uyumluluk değerinin ölçümlenmesi ve jenerasyonun ilerletilmesini sağlayan fonksiyonun çağrılması gibi rolleri üstlenir. Yukarıda belirtilen ana metot **DE/rand/1** mutasyon fonksiyonu kullanmaktadır.

```java
//This method is used for iteratively pushing the generations in the differential evolution (DE/rand/1) further. Defined as a separate function to increase modul
public static ArrayList<DEIndividual> nextGenerationContinuousOptimization_DErand1(Problem problem, ArrayList<DEIndividual> population,
                                                                                    double F, double crossoverRate) {

    //evaluate the fitness of the population
    population = DEFitness.evaluateDEPopulationFitness(problem, population);

    //show the average fitness of the population
    DEFitness.showAverageDEPopulationFitness(problem, population);

    //mutate the individuals with mutation method de/rand/1
    ArrayList<DEIndividual> mutatedIndividuals = DECrossover.mutateDEPopulation_DErand1(problem, population, F, crossoverRate);

    //return mutated individuals
    return mutatedIndividuals;
}
```

DE/rand/1 jenerasyon ilerletme metotu yukarıda görüldüğü gibidir. Popülasyonla ilgili uyumluluk ölçümleme ve mutasyona uğratma gibi faaliyetleri gerçekleştiren metotların çağrılması yoluyla jenerasyon ilerletilir.

```java
//This method is used for iteratively pushing the generations in the differential evolution (DE/rand/2) further. Defined as a separate function to increase
public static ArrayList<DEIndividual> nextGenerationContinuousOptimization_DErand2(Problem problem, ArrayList<DEIndividual> population,
                                                                                    double F, double crossoverRate) {

    //evaluate the fitness of the population
    population = DEFitness.evaluateDEPopulationFitness(problem, population);

    //show the average fitness of the population
    DEFitness.showAverageDEPopulationFitness(problem, population);

    //mutate the individuals with mutation method de/rand/2
    ArrayList<DEIndividual> mutatedIndividuals = DECrossover.mutateDEPopulation_DErand2(problem, population, F, crossoverRate);

    //return the mutated individuals
    return mutatedIndividuals;
}
```

DE/rand/2 jenerasyon ilerletme metotu

```java
//This method is used for iteratively pushing the generations in the differential evolution (DE/best/1) further. Defined as a separate function to
public static ArrayList<DEIndividual> nextGenerationContinuousOptimization_DEbest1(Problem problem, ArrayList<DEIndividual> population,
                                                                                    double F, double crossoverRate) {

    //evalute the fitness of the population
    population = DEFitness.evaluateDEPopulationFitness(problem, population);

    //show the average fitness of the population
    DEFitness.showAverageDEPopulationFitness(problem, population);

    //mutate the individuals with mutation method de/best/1
    ArrayList<DEIndividual> mutatedIndividuals = DECrossover.mutateDEPopulation_DEbest1(problem, population, F, crossoverRate);

    //return the mutated individuals
    return mutatedIndividuals;
}
```

DE/best/1 jenerasyon ilerletme metotu

```java
//This method is used for iteratively pushing the generations in the differential evolution (DE/best/2) further. Defined as a separate function to increase modu
public static ArrayList<DEIndividual> nextGenerationContinuousOptimization_DEbest2(Problem problem, ArrayList<DEIndividual> population,
                                                                                    double F, double crossoverRate) {

    //evaluate the fitness of the population
    population = DEFitness.evaluateDEPopulationFitness(problem, population);

    //show the average fitness level of the population
    DEFitness.showAverageDEPopulationFitness(problem, population);

    //mutate the individuals with mutation method de/best/2
    ArrayList<DEIndividual> mutatedIndividuals = DECrossover.mutateDEPopulation_DEbest2(problem, population, F, crossoverRate);

    //return mutated individuals
    return mutatedIndividuals;
}
```

DE/best/2 jenerasyon ilerletme metotu

```java
//This method is used for iteratively pushing the generations in the differential evolution (DE/current_to_best/1) further. Defined as a separate function
public static ArrayList<DEIndividual> nextGenerationContinuousOptimization_DEcurToBest1(Problem problem, ArrayList<DEIndividual> population,
                                                                                         double F, double crossoverRate) {

    //evaluate the fitness of the population
    population = DEFitness.evaluateDEPopulationFitness(problem, population);

    //show the average fitness of the population
    DEFitness.showAverageDEPopulationFitness(problem, population);

    //mutate the individuals with mutation method de/current_to_best/1
    ArrayList<DEIndividual> mutatedIndividuals = DECrossover.mutateDEPopulation_DEcurToBest1(problem, population, F, crossoverRate);

    //return mutated individuals
    return mutatedIndividuals;
}
```

DE/current_to_best/1 jenerasyon ilerletme metotu

```java
//This is the main wrapper function to be able to use for running the differential evolution (DE/rand/2) with predefined hyper-parameters
public static ArrayList<DEIndividual> runContinuousOptimization_DErand2(Problem problem, int populationSize, int maxGenerations,
                                                                         double F, double crossoverRate) {

    //generate initial population
    ArrayList<DEIndividual> population = DEPopulation.generateDEPopulation(problem, populationSize);

    //evaluate initial population fitness
    DEFitness.evaluateDEPopulationFitness(problem, population);

    //show average initial population fitness
    DEFitness.showAverageDEPopulationFitness(problem, population);

    //run the algorithm for maxGenerations times
    System.out.println("GENERATION 0 : \n");

    for(int i = 0; i<maxGenerations; i++) {

    System.out.println("GENERATION :" + i + " \n");

    ArrayList<DEIndividual> nextPopulation = nextGenerationContinuousOptimization_DErand2(problem, population, F, crossoverRate);

    System.out.println("\n_____");

    population = nextPopulation;
    }

    //evaluate the final population fitness
    population = DEFitness.evaluateDEPopulationFitness(problem, population);

    //return final population
    return population;
}
```

DE/rand/2 ana operasyon metotu

```java
//This is the main wrapper function to be able to use for running the differential evolution (DE/best/1) with predefined hyper-parameters
public static ArrayList<DEIndividual> runContinuousOptimization_DEbest1(Problem problem, int populationSize, int maxGenerations,
        double F, double crossoverRate) {


    //generate initial population
    ArrayList<DEIndividual> population = DEPopulation.generateDEPopulation(problem, populationSize);

    //evaluate initial population fitness
    DEFitness.evaluateDEPopulationFitness(problem, population);

    //show initial population fitness
    DEFitness.showAverageDEPopulationFitness(problem, population);


    //Run the algorithm for maxGenerations times
    System.out.println("GENERATION 0 : \n");

    for(int i = 0; i<maxGenerations; i++) {

    System.out.println("GENERATION :" + i + " \n");

    ArrayList<DEIndividual> nextPopulation = nextGenerationContinuousOptimization_DEbest1(problem, population, F, crossoverRate);

    System.out.println("\n_____");

    population = nextPopulation;
    }

    //evaluate the fitness for the final population
    population = DEFitness.evaluateDEPopulationFitness(problem, population);

    //return the final population
    return population;
}
```

DE/best/1 ana operasyon metotu

```java
//This is the main wrapper function to be able to use for running the differential evolution (DE/best/2) with predefined hyper-parameters
public static ArrayList<DEIndividual> runContinuousOptimization_DEbest2(Problem problem, int populationSize, int maxGenerations,
        double F, double crossoverRate) {

    //generate initial population
    ArrayList<DEIndividual> population = DEPopulation.generateDEPopulation(problem, populationSize);

    //evaluate initial population fitness
    DEFitness.evaluateDEPopulationFitness(problem, population);

    //show average initial population fitness
    DEFitness.showAverageDEPopulationFitness(problem, population);


    //Run the algorithm for maxGenerations times.
    System.out.println("GENERATION 0 : \n");

    for(int i = 0; i<maxGenerations; i++) {

    System.out.println("GENERATION :" + i + " \n");

    ArrayList<DEIndividual> nextPopulation = nextGenerationContinuousOptimization_DEbest2(problem, population, F, crossoverRate);

    System.out.println("\n_____");

    population = nextPopulation;
    }

    //evaluate the final population
    population = DEFitness.evaluateDEPopulationFitness(problem, population);

    //return the final population
    return population;
}
```

DE/best/2 ana operasyon metotu

```java
//This is the main wrapper function to be able to use for running the differential evolution (DE/current_to_best/1) with predefined hyper-parameters
public static ArrayList<DEIndividual> runContinuousOptimization_DEcurToBest1(Problem problem, int populationSize, int maxGenerations,
        double F, double crossoverRate) {


    //Generate initial population
    ArrayList<DEIndividual> population = DEPopulation.generateDEPopulation(problem, populationSize);

    //Evaluate the initial population fitness
    DEFitness.evaluateDEPopulationFitness(problem, population);

    //Show the average population fitness
    DEFitness.showAverageDEPopulationFitness(problem, population);


    //Run the algorithm for maxGenerations times
    System.out.println("GENERATION 0 : \n");

    for(int i = 0; i<maxGenerations; i++) {

    System.out.println("GENERATION :" + i + " \n");

    ArrayList<DEIndividual> nextPopulation = nextGenerationContinuousOptimization_DEcurToBest1(problem, population, F, crossoverRate);

    System.out.println("\n_____");

    population = nextPopulation;
    }

    //evaluate the fitness of the final population
    population = DEFitness.evaluateDEPopulationFitness(problem, population);

    //return the final population
    return population;
}
```

DE/current_to_best/1 ana operasyon metotu

2.Yardımcı Metotlar

```java
package support;

import java.util.ArrayList;


public class MathSupport {

    /*
     *  THIS CLASS CONTAINS THE MATHEMATICAL SUPPORT FUNCTIONS THAT THE PROGRAM USES FOR EVALUATING METRICS FOR SOLUTIONS.
     *
     */


    //This method calculates the standard deviation of the given elements in an arraylist
    public static double sd (ArrayList<Double> results)
    {

        double average = average(results);
        double temp = 0;

        for (int i = 0; i < results.size(); i++)
        {
            double val = results.get(i);

            double squrDiffToMean = Math.pow(val - average, 2);

            temp += squrDiffToMean;
        }

        double meanOfDiffs = (double) temp / (double) (results.size());

        return Math.sqrt(meanOfDiffs);
    }


    //This method calculates the median of the given elements in an arraylist
    public static double median (ArrayList<Double> results) {

        double median = (results.get(results.size()/2) + results.get(results.size()/2 - 1))/2;
        return median;
    }


    //This method takes the average of the given elements in an arraylist
    public static double average (ArrayList<Double> results) {

        double summation = 0;
        for (int i = 0; i<results.size(); i++) {
            summation = summation + results.get(i);
        }

        double average = summation/results.size();
        return average;
    }
```

Yukarıda projenin gerçekleştirilmesi için kullanılan yardımcı sınıfların biri olan MathSupport sınıfı görüntülenmektedir. Bu sınıfın içerisinde çözümlerle ilgili metriklerin hesaplanmasında kullanılan çeşitli fonksiyonlar bulunmaktadır. Yukarıda bu fonksiyonlara örnek olarak sd, standart sapmanın hesaplanması için kullanılmaktayken; median, medyan değerin hesaplanması için, average ortalama değerin hesaplanması için kullanılmaktadır.

```java
//This method takes the best result in a given arraylist and mainly used for taking the maximum fitness value in a metrics array.
public static double getBestResult(ArrayList<Double> results){

    double max = 0;
    for(int i = 0; i<results.size(); i++) {

        if(results.get(i) > max) {
            max = results.get(i);
        }
        else {
            continue;
        }
    }

    return max;
}


//This method takes the worst result in a given arraylist and mainly used for taking the minimum fitness value in a metrics array.
public static double getWorstResult(ArrayList<Double> results){

    double min = results.get(0);
    for(int i = 0; i<results.size(); i++) {

        if(results.get(i) < min) {
            min = results.get(i);
        }

    }

    return min;
}
```

Metrikler arasından en kötü ve en iyi sonuçlara erişilmesini sağlayan yardımcı fonksiyonlar da yukarıda görüntülendiği gibidir.

```java
package support;

import java.util.ArrayList;

public class PrintSupport {

    /*
     *
     *  THIS CLASS CONTAINS THE METHODS THAT IS USED FOR PRINTING THE METRICS FOR DIFFERENT SETS OF CALCULATIONS WITH DIFFERENT PARAMETERS
     *
     * */


    //This method prints the multi-metrics for an increased number of calculations with different parameter sets.
    public static void printMultiMetrics(Problem problem, ArrayList<ArrayList<Double>> multiResultsAvg, ArrayList<ArrayList<Double>> multiMetrics) {

        for(int i = 0; i<multiResultsAvg.size(); i++) {

            System.out.println("RESULTS FOR PARAMETER SET - "+i+" : ");

            ArrayList<Double> resultsAvg = multiResultsAvg.get(i);

            for(int j = 0; j<resultsAvg.size(); j++ ) {
                System.out.println("RUN "+ (j+1) + " : " + resultsAvg.get(j)  + " / "+ problem.getCity().getCustomerAmount() + " ( %" + (resultsAvg.get(j)/problem.getC
            }
            System.out.println("_____________________");

            ArrayList<Double> metrics = multiMetrics.get(i);


            System.out.println("AVERAGE : " + metrics.get(0)  + " / "+ problem.getCity().getCustomerAmount() + " ( %" + (metrics.get(0)/problem.getCity().getCustomerAm
            System.out.println("STANDARD DEVIATION : "+ metrics.get(1) + " / "+ problem.getCity().getCustomerAmount() + " ( %" + (metrics.get(1)/problem.getCity().getC
            System.out.println("BEST RESULT : "+ metrics.get(2)  + " / "+ problem.getCity().getCustomerAmount() + " ( %" + (metrics.get(2)/problem.getCity().getCustome
            System.out.println("WORST RESULT : " + metrics.get(3)  + " / "+ problem.getCity().getCustomerAmount() + " ( %" + (metrics.get(3)/problem.getCity().getCustor
            System.out.println("MEDIAN : " + metrics.get(4)  + " / "+ problem.getCity().getCustomerAmount() + " ( %" + (metrics.get(4)/problem.getCity().getCustomerAmo

            System.out.println("**********************");
            System.out.println();


            }

    }
}
```

Bir diğer yardımcı sınıf olan PrintSupport ise, kullanıcı dostu olacak şekilde çözüm metriklerini konsol ekranında görüntülemek amacıyla dizayn edilmiş metotları içermektedir. Görüntüde yer alan metot; çözümlere ait ortalama, standart sapma, en iyi ve en kötü sonuç ve medyan sonuç gibi çeşitli bilgilerin ekranda görüntülenmesini sağlamaktadır.

Diğer Bağımsız Test Sınıfları :

- **COTest_DErand1.java** : Sadece DE/rand/1 mutasyon tipini kullanarak problemi çözmeye odaklanan test sınıfıdır.
- **COTest_DErand2.java** : Sadece DE/rand/2 mutasyon tipini kullanarak problemi çözmeye odaklanan test sınıfıdır.
- **COTest_DEbest1.java** : Sadece DE/best/1 mutasyon tipini kullanarak problemi çözmeye odaklanan test sınıfıdır.
- **COTest_DEbest2.java** : Sadece DE/best/2 mutasyon tipini kullanarak problemi çözmeye odaklanan test sınıfıdır.
- **COTest_DEcurToBest1.java** : Sadece De/current_to_best/1 mutasyon tipini kullanarak problemi çözmeye odaklanan test sınıfıdır.
- **UnitTests.java :** Çeşitli mikro ve makro geliştirme öğelerinin ve modellerin test edildiği sınıftır.

3.Kod Ekran Görüntüleri

(SÜREKLİ OPTİMİZASYON PAKETİ)

```
public class DEPopulation {

    /*
     *
     * THIS CLASS IS USED FOR GENERATING RANDOMIZED POPULATIONS FOR DIFFERENTIAL EVOLUTION ALGORITHM AND PRINTING SPECIFIC DATA ABOUT THEM
     *
     */


    //this method is used for generating an initial population for the differential evolution algorithm
    public static ArrayList<DEIndividual> generateDEPopulation(Problem problem, int populationSize){

        ArrayList<DEIndividual> population = new ArrayList<DEIndividual>();
        for(int i = 0; i<populationSize; i++) {
            DEIndividual individual = new DEIndividual(problem);
            population.add(individual);
        }

        return population;
    }


    //this method is used for showing the general data about the population
    public static void showDEPopulationData(ArrayList<DEIndividual> population) {

        System.out.println();
        System.out.println("PRINTING THE INFORMATION OF THE POPULATION ... ");
        System.out.println();
        for(int i = 0; i<population.size(); i++) {
            DEIndividual iv = population.get(i);
            System.out.println("INDIVIDUAL NUMBER: " + i);
            System.out.println("INDIVIDUAL FITNESS SCORE: " + iv.getFitness());
            iv.showChromosome();
            System.out.println("_________________");
        }
    }

}
```

```java
public class DEIndividual implements Comparable<DEIndividual> {


    /*
     *
     * THIS CLASS IS USED FOR DEFINING THE INDIVIDUAL OBJECT, WHICH IS THE MAIN POPULATION ACTOR IN DIFFERENTIAL EVOLUTION ALGORITHM
     *
     */


    private int id;
    private Problem problem;
    private ArrayList<Base> chromosome;
    private Double fitness;

    private static int counter = 0;

    public DEIndividual(Problem problem) {
        counter++;
        this.id = counter;
        this.problem = problem;
        this.chromosome = randomizeChromosome();
        this.fitness = 0.0;
    }


    //this method is used for randomizing the chromosome (base coordinates) of the individual on creation
    public ArrayList<Base> randomizeChromosome(){

        Random r = new Random();
        double x;
        double y;
        ArrayList<Base> bases = new ArrayList<Base>();
        if(problem.getCity().getBaseLocationAmount() == 0) {

            for(int i = 0; i<problem.getCity().getBaseAmount(); i++) {

                //create random locations in a continous space if the city has no predefined base locations
                x = problem.getCity().getWidth() * r.nextDouble();
                y = problem.getCity().getHeight() * r.nextDouble();
                Base base = new Base(x, y, problem.getCoverRadius());
                bases.add(base);
            }

            return bases;
        }
        else {

            for(BaseLocation location  : problem.getBaseLocations()) {
                location.setOccupied(false);
            }

            int choice;
            for(int i = 0; i<problem.getCity().getBaseAmount(); i++) {

                choice = r.nextInt(problem.getBaseLocations().size());
                if(problem.getBaseLocations().get(choice).isOccupied() == true) {
                    i = i -1;
                    continue;
                }
                else {

                    //select from predefined base locations if they exist
                    x = problem.getBaseLocations().get(choice).getX();
                    y = problem.getBaseLocations().get(choice).getY();
                    Base base = new Base(x, y, problem.getCoverRadius());
```

```
            else {

                //select from predefined base locations if they exist
                x = problem.getBaseLocations().get(choice).getX();
                y = problem.getBaseLocations().get(choice).getY();
                Base base = new Base(x, y, problem.getCoverRadius());
                bases.add(base);
                problem.getBaseLocations().get(choice).setOccupied(true);
            }
        }

        return bases;
    }
}


//this method is used for printing the chromosome information (base coordintes) of the individual
public void showChromosome() {
    System.out.println();
    System.out.println("\n INDIVIDUAL BASE COORDINATES : ");
    for(int i = 0; i<chromosome.size(); i++) {
        Base base = chromosome.get(i);
        System.out.print("Base "+i+" ("+
                         String.format("%.2f", base.getX())+
                         " / "+
                         String.format("%.2f", base.getY())+
                         ") , ");
    }
    System.out.println();
}

public static int getIndividualAmount() {
    return counter;
}

public Problem getProblem() {
    return problem;
}

public void setProblem(Problem problem) {
    this.problem = problem;
}

public ArrayList<Base> getChromosome() {
    return chromosome;
}

public void setChromosome(ArrayList<Base> chromosome) {
    this.chromosome = chromosome;
}

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public Double getFitness() {
    return fitness;
}

public void setFitness(Double fitness) {
    this.fitness = fitness;
}
```

```java
public class DEFitness {


    /*
     *
     * THIS CLASS IS USED FOR EVALUATING THE FITNESS LEVEL OF THE INDIVIDUALS FOR DIFFERENTIAL EVOLUTION ALGORITHM
     *
     */



    //this method is used for evaluating the fitness score of a single individual
    public static double evaluateDEIndividualFitness(Problem problem, DEIndividual iv) {

        ArrayList<Customer> customers = problem.getCustomers();

        double fitness = 0;
        for(int j = 0; j<customers.size(); j++) {

            Customer customer = customers.get(j);
            for(int k = 0; k<iv.getChromosome().size(); k++) {

                Base base = iv.getChromosome().get(k);

                //take distance and coverage status of each customer in the city.
                double deltaXPow = Math.pow(customer.getX() - base.getX(), 2);
                double deltaYPow = Math.pow(customer.getY() - base.getY(), 2);
                double sqrtSum = Math.sqrt((deltaXPow + deltaYPow));

                //evaluate fitnesss score
                if(sqrtSum <= problem.getCoverRadius()){
                    fitness = fitness + 1;
                    break;
                }else {
                    continue;
                }
            }
        }

        iv.setFitness(fitness);

        return fitness;
    }


    //this method is used for evaluating the fitness score of the entire differential evoliton population arraylist
    public static ArrayList<DEIndividual> evaluateDEPopulationFitness(Problem problem, ArrayList<DEIndividual> population){

        for(int i = 0; i<population.size(); i++) {

            DEIndividual iv = population.get(i);

            //evaluate the fitness of each individual
            double fitness = evaluateDEIndividualFitness(problem, iv);


            //set fitness score of each individual
            iv.setFitness(fitness);
        }

        return population;
    }







    //this method is used for showing the population scores of the each individual in the differential evolution population
    public static void showDEPopulationFitnessScores(Problem problem, ArrayList<DEIndividual> population) {

        System.out.println();
        System.out.println("SHOWING POPULATION FITNESS SCORES ...");
        System.out.println();
        for(int i = 0; i<population.size(); i++) {
            double prc = (population.get(i).getFitness()/problem.getCity().getCustomerAmount())*100;
            System.out.println("INDIVIDUAL "+i+" FITNESS SCORE : " + population.get(i).getFitness() + " / "+ problem.getCity().getCustomerAmount() + " ( %" + prc + " )
        }
        System.out.println();
    }


    //this method is used for getting the average population fitenss score in the given population
    public static double getAverageDEPopulationFitness(ArrayList<DEIndividual> population) {

        double summation = 0;
        for(int i = 0; i<population.size(); i++) {

            summation = summation + population.get(i).getFitness();
        }

        double average = (double) (summation/population.size());

        return average;
    }


    //this method is used for printing the average population fitness score in the given population
    public static void showAverageDEPopulationFitness(Problem problem, ArrayList<DEIndividual> population) {

        double summation = 0;
        for(int i = 0; i<population.size(); i++) {

            summation = summation + population.get(i).getFitness();
        }

        double average = summation/population.size();

        double prc = (average/problem.getCity().getCustomerAmount())*100;
        System.out.println("AVERAGE FITNESS OF POPULATION : "+average+" / "+problem.getCity().getCustomerAmount() + " ( %" + prc + " )");
    }

}
```

```java
public class DECrossover {


    /*
     *
     * THIS CLASS IS USED FOR APPLYING MUTATION TO THE DIFFERENTIAL EVOLUTION POPULATION BASED ON PARAMETERS
     *
     */



    //this method is used for mutating the differential evolution population with the specified method : DE/rand/1
    public static ArrayList<DEIndividual> mutateDEPopulation_DErand1(Problem problem, ArrayList<DEIndividual> population, double F, double crossoverRate){

        Random r = new Random();

        ArrayList<DEIndividual> mutatedPopulation = new ArrayList<DEIndividual>();
        for (int i = 0; i<population.size(); i++) {

            DEIndividual deiv = population.get(i);
            int r1Idx = -1;
            int r2Idx = -1;
            int r3Idx = -1;

            //specify indexes for the selected population individuals
            int idx = r.nextInt(population.size());
            while(idx == i)
                idx = r.nextInt(population.size());
            r1Idx = idx;
            while(idx == i || idx == r1Idx)
                idx = r.nextInt(population.size());
            r2Idx = idx;
            while(idx == i || idx == r1Idx || idx == r2Idx)
                idx = r.nextInt(population.size());
            r3Idx = idx;

            DEIndividual r1 = population.get(r1Idx);
            DEIndividual r2 = population.get(r2Idx);
            DEIndividual r3 = population.get(r3Idx);

            //mutate the individuals
            DEIndividual newIndividual = DECrossover.mutateDEIndividual_DErand1(problem, deiv, r1, r2, r3, F, crossoverRate);
            mutatedPopulation.add(newIndividual);
        }

        //return mutated individual
        return mutatedPopulation;
    }


    //this method is used for mutating the differential evolution population with the specified method : DE/rand/2
    public static ArrayList<DEIndividual> mutateDEPopulation_DErand2(Problem problem, ArrayList<DEIndividual> population, double F, double crossoverRate){


        Random r = new Random();

        ArrayList<DEIndividual> mutatedPopulation = new ArrayList<DEIndividual>();
        for (int i = 0; i<population.size(); i++) {

            DEIndividual deiv = population.get(i);
            int r1Idx = -1;
            int r2Idx = -1;
            int r3Idx = -1;
            int r4Idx = -1;
            int r5Idx = -1;

            //specify indexes for the selected population individuals
            int idx = r.nextInt(population.size());
            while(idx == i)
```

```java
    //this method is used for mutating the differential evolution population with the specified method : DE/rand/2
    public static ArrayList<DEIndividual> mutateDEPopulation_DErand2(Problem problem, ArrayList<DEIndividual> population, double F, double crossoverRate){


        Random r = new Random();

        ArrayList<DEIndividual> mutatedPopulation = new ArrayList<DEIndividual>();
        for (int i = 0; i<population.size(); i++) {

            DEIndividual deiv = population.get(i);
            int r1Idx = -1;
            int r2Idx = -1;
            int r3Idx = -1;
            int r4Idx = -1;
            int r5Idx = -1;

            //specify indexes for the selected population individuals
            int idx = r.nextInt(population.size());
            while(idx == i)
                idx = r.nextInt(population.size());
            r1Idx = idx;
            while(idx == i || idx == r1Idx)
                idx = r.nextInt(population.size());
            r2Idx = idx;
            while(idx == i || idx == r1Idx || idx == r2Idx)
                idx = r.nextInt(population.size());
            r3Idx = idx;
            while(idx == i || idx == r1Idx || idx == r2Idx || idx == r3Idx)
                idx = r.nextInt(population.size());
            r4Idx = idx;
            while(idx == i || idx == r1Idx || idx == r2Idx || idx == r3Idx || idx == r4Idx  )
                idx = r.nextInt(population.size());
            r5Idx = idx;

            DEIndividual r1 = population.get(r1Idx);
            DEIndividual r2 = population.get(r2Idx);
            DEIndividual r3 = population.get(r3Idx);
            DEIndividual r4 = population.get(r4Idx);
            DEIndividual r5 = population.get(r5Idx);

            //mutate the individuals
            DEIndividual newIndividual = DECrossover.mutateDEIndividual_DErand2(problem, deiv, r1, r2, r3, r4, r5, F, crossoverRate);
            mutatedPopulation.add(newIndividual);
        }

        //return mutated population
        return mutatedPopulation;
    }


    //this method is used for mutating the differential evolution population with the specified method : DE/best/1
    public static ArrayList<DEIndividual> mutateDEPopulation_DEbest1(Problem problem, ArrayList<DEIndividual> population, double F, double crossoverRate){

        Random r = new Random();

        DEIndividual bestDeiv = getBestDEIndividual(population);
        int bestDeivIdx = -1;
        for (int i = 0; i<population.size(); i++) {
            if( population.get(i) == bestDeiv)
                bestDeivIdx = i;
        }

        ArrayList<DEIndividual> mutatedPopulation = new ArrayList<DEIndividual>();
        for (int i = 0; i<population.size(); i++) {

            DEIndividual deiv = population.get(i);
            int r1Idx = -1;
            int r2Idx = -1;
```

```
            int r4Idx = -1;


            //specify different indexes for population individuals
            int idx = r.nextInt(population.size());
            while(idx == i || idx == bestDeivIdx )
                idx = r.nextInt(population.size());
            r1Idx = idx;
            while(idx == i || idx == bestDeivIdx || idx == r1Idx )
                idx = r.nextInt(population.size());
            r2Idx = idx;
            while(idx == i || idx == bestDeivIdx || idx == r1Idx || idx == r2Idx  )
                idx = r.nextInt(population.size());
            r3Idx = idx;
            while(idx == i || idx == bestDeivIdx || idx == r1Idx || idx == r2Idx || idx == r3Idx)
                idx = r.nextInt(population.size());
            r4Idx = idx;

            DEIndividual r1 = population.get(r1Idx);
            DEIndividual r2 = population.get(r2Idx);
            DEIndividual r3 = population.get(r3Idx);
            DEIndividual r4 = population.get(r4Idx);

            //mutate each individual
            DEIndividual newIndividual = DECrossover.mutateDEIndividual_DEbest2(problem, deiv, bestDeiv, r1, r2, r3, r4, F, crossoverRate);
            mutatedPopulation.add(newIndividual);
        }

        //return mutated population
        return mutatedPopulation;
    }


    //this method is used for mutating the differential evolution population with the specified method : DE/current_to_best/1
    public static ArrayList<DEIndividual> mutateDEPopulation_DEcurToBest1(Problem problem, ArrayList<DEIndividual> population, double F, double crossoverRate){


        Random r = new Random();

        DEIndividual bestDeiv = getBestDEIndividual(population);
        int bestDeivIdx = -1;
        for (int i = 0; i<population.size(); i++) {
            if( population.get(i) == bestDeiv)
                bestDeivIdx = i;
        }

        ArrayList<DEIndividual> mutatedPopulation = new ArrayList<DEIndividual>();
        for (int i = 0; i<population.size(); i++) {

            DEIndividual deiv = population.get(i);
            int r1Idx = -1;
            int r2Idx = -1;

            //specify different indexes for population individuals
            int idx = r.nextInt(population.size());
            while(idx == i || idx == bestDeivIdx )
                idx = r.nextInt(population.size());
            r1Idx = idx;
            while(idx == i || idx == bestDeivIdx || idx == r1Idx )
                idx = r.nextInt(population.size());
            r2Idx = idx;

            DEIndividual r1 = population.get(r1Idx);
            DEIndividual r2 = population.get(r2Idx);
```

```java
//this method is used for mutating a single individual inside the population with DE/rand/1 method
public static DEIndividual mutateDEIndividual_DErand1(Problem problem,
                                                       DEIndividual deiv,
                                                       DEIndividual r1,
                                                       DEIndividual r2,
                                                       DEIndividual r3,
                                                       double F,
                                                       double crossoverRate) {


    DEIndividual mutatedIndividual = new DEIndividual(problem);

    ArrayList<Base> mutatedChromosome = new ArrayList<Base>();

    Random r = new Random();
    crossoverRate = crossoverRate * 100;

    //define the mutaton index
    int mutationIndex = r.nextInt(deiv.getChromosome().size());


    for (int i = 0; i<deiv.getChromosome().size(); i++) {

        int coDie = r.nextInt(100);

        //create the mutant vectors
        double mutantX = r1.getChromosome().get(i).getX() + (F * (r2.getChromosome().get(i).getX() - r3.getChromosome().get(i).getX()));
        double mutantY = r1.getChromosome().get(i).getY() + (F * (r2.getChromosome().get(i).getY() - r3.getChromosome().get(i).getY()));

        double trialX;
        double trialY;

        //create the trial vector
        if (coDie < crossoverRate || i == mutationIndex) {

            trialX = mutantX;
            trialY = mutantY;

        }
        else {

            trialX = deiv.getChromosome().get(i).getX();
            trialY = deiv.getChromosome().get(i).getY();

        }

        //create the mutated base and add to chromosome
        Base mutatedBase = new Base(trialX, trialY, problem.getCoverRadius());

        mutatedChromosome.add(mutatedBase);
    }

    mutatedIndividual.setChromosome(mutatedChromosome);

    double newFitness = DEFitness.evaluateDEIndividualFitness(problem, mutatedIndividual);
    double oldFitness = deiv.getFitness();

    //if the new fitness is better, save it
    if(newFitness <= oldFitness) {
        mutatedIndividual = deiv;
    }
    else {
        mutatedIndividual.setChromosome(mutatedChromosome);
    }

    //return mutated individual
    return mutatedIndividual;
```

```java
    //this method is used for mutating a single individual in the population with DE/rand/2 method
    public static DEIndividual mutateDEIndividual_DErand2(Problem problem,
                                                           DEIndividual deiv,
                                                           DEIndividual r1,
                                                           DEIndividual r2,
                                                           DEIndividual r3,
                                                           DEIndividual r4,
                                                           DEIndividual r5,
                                                           double F,
                                                           double crossoverRate) {

        DEIndividual mutatedIndividual = new DEIndividual(problem);

        ArrayList<Base> mutatedChromosome = new ArrayList<Base>();

        Random r = new Random();
        crossoverRate = crossoverRate * 100;

        int mutationIndex = r.nextInt(deiv.getChromosome().size());


        for (int i = 0; i<deiv.getChromosome().size(); i++) {

            int coDie = r.nextInt(100);

            //define the mutant vectors
            double mutantX = r1.getChromosome().get(i).getX() + (F * (r2.getChromosome().get(i).getX() - r3.getChromosome().get(i).getX()) +
                            F * (r4.getChromosome().get(i).getX() - r5.getChromosome().get(i).getX()));


            double mutantY = r1.getChromosome().get(i).getY() + (F * (r2.getChromosome().get(i).getY() - r3.getChromosome().get(i).getY()) +
                    F * (r4.getChromosome().get(i).getY() - r5.getChromosome().get(i).getY()));

            double trialX;
            double trialY;


            //create the trial vectors
            if (coDie < crossoverRate || i == mutationIndex) {

                trialX = mutantX;
                trialY = mutantY;

            }
            else {

                trialX = deiv.getChromosome().get(i).getX();
                trialY = deiv.getChromosome().get(i).getY();

            }

            //create the mutated base and add it to the chromosome
            Base mutatedBase = new Base(trialX, trialY, problem.getCoverRadius());

            mutatedChromosome.add(mutatedBase);
        }

        mutatedIndividual.setChromosome(mutatedChromosome);

        double newFitness = DEFitness.evaluateDEIndividualFitness(problem, mutatedIndividual);
        double oldFitness = deiv.getFitness();

        //if the new fitness is better, save it
        if(newFitness <= oldFitness) {
            mutatedIndividual = deiv;
        }
        else {
            mutatedIndividual.setChromosome(mutatedChromosome);
```

```java
//this method is used for mutating a single individual with DE/best/1 method
public static DEIndividual mutateDEIndividual_DEbest1(Problem problem,
                                                       DEIndividual deiv,
                                                       DEIndividual bestDeiv,
                                                       DEIndividual r1,
                                                       DEIndividual r2,
                                                       double F,
                                                       double crossoverRate) {


    DEIndividual mutatedIndividual = new DEIndividual(problem);

    ArrayList<Base> mutatedChromosome = new ArrayList<Base>();

    Random r = new Random();
    crossoverRate = crossoverRate * 100;

    int mutationIndex = r.nextInt(deiv.getChromosome().size());


    for (int i = 0; i<deiv.getChromosome().size(); i++) {

        int coDie = r.nextInt(100);

        //create the mutant vectors
        double mutantX = bestDeiv.getChromosome().get(i).getX() + (F * (r1.getChromosome().get(i).getX() - r2.getChromosome().get(i).getX()));
        double mutantY = bestDeiv.getChromosome().get(i).getY() + (F * (r1.getChromosome().get(i).getY() - r2.getChromosome().get(i).getY()));

        double trialX;
        double trialY;


        //create the trial vectors
        if (coDie < crossoverRate || i == mutationIndex) {

            trialX = mutantX;
            trialY = mutantY;

        }
        else {

            trialX = deiv.getChromosome().get(i).getX();
            trialY = deiv.getChromosome().get(i).getY();

        }

        //create the mutated base and add it to the chromosome
        Base mutatedBase = new Base(trialX, trialY, problem.getCoverRadius());

        mutatedChromosome.add(mutatedBase);
    }

    mutatedIndividual.setChromosome(mutatedChromosome);

    double newFitness = DEFitness.evaluateDEIndividualFitness(problem, mutatedIndividual);
    double oldFitness = deiv.getFitness();

    //if the new fitness is better, save it
    if(newFitness <= oldFitness) {
        mutatedIndividual = deiv;
    }
    else {
        mutatedIndividual.setChromosome(mutatedChromosome);
    }
```

```java
    //this method is used for applying mutation to a single individual with DE/current_to_best/1 method
    public static DEIndividual mutateDEIndividual_DEcurToBest1(Problem problem,
                                                                DEIndividual deiv,
                                                                DEIndividual bestDeiv,
                                                                DEIndividual r1,
                                                                DEIndividual r2,
                                                                double F,
                                                                double crossoverRate) {


        DEIndividual mutatedIndividual = new DEIndividual(problem);

        ArrayList<Base> mutatedChromosome = new ArrayList<Base>();

        Random r = new Random();
        crossoverRate = crossoverRate * 100;

        int mutationIndex = r.nextInt(deiv.getChromosome().size());


        for (int i = 0; i<deiv.getChromosome().size(); i++) {

            int coDie = r.nextInt(100);

            //create the mutant vectors
            double mutantX = deiv.getChromosome().get(i).getX() + (F * (bestDeiv.getChromosome().get(i).getX() - deiv.getChromosome().get(i).getX()) +
                    (F * (r1.getChromosome().get(i).getX() - r2.getChromosome().get(i).getX())));
            double mutantY = deiv.getChromosome().get(i).getY() + (F * (bestDeiv.getChromosome().get(i).getY() - deiv.getChromosome().get(i).getY()) +
                    (F * (r1.getChromosome().get(i).getY() - r2.getChromosome().get(i).getY())));

            double trialX;
            double trialY;


            //create the trial vectors
            if (coDie < crossoverRate || i == mutationIndex) {

                trialX = mutantX;
                trialY = mutantY;

            }
            else {

                trialX = deiv.getChromosome().get(i).getX();
                trialY = deiv.getChromosome().get(i).getY();

            }

            //create the mutated base and save it to the chromosome
            Base mutatedBase = new Base(trialX, trialY, problem.getCoverRadius());

            mutatedChromosome.add(mutatedBase);
        }

        mutatedIndividual.setChromosome(mutatedChromosome);

        double newFitness = DEFitness.evaluateDEIndividualFitness(problem, mutatedIndividual);
        double oldFitness = deiv.getFitness();

        //if the new fitness is better, save it
        if(newFitness <= oldFitness) {
            mutatedIndividual = deiv;
        }
        else {
            mutatedIndividual.setChromosome(mutatedChromosome);
```

```java
    //this method is used for achieving the individual with the best fitness level in a population
    public static DEIndividual getBestDEIndividual(ArrayList<DEIndividual> population) {

        DEIndividual maxDeiv = population.get(0);
        double max = 0;
        for(int i = 0; i<population.size(); i++) {

            //find the individual with the maximum fitness level
            if(population.get(i).getFitness() > max) {
                max = population.get(i).getFitness();
                maxDeiv = population.get(i);
            }
        }

        //return individual with the maximum fitness score
        return maxDeiv;
    }
}
```

(GENETİK ALGORİTMA PAKETİ)

```java
public class Fitness {

    /*
     *
     * THIS CLASS IS USED FOR FITNESS EVALUATION AND REPRESENTATION OF THE GENETIC ALGORITHM POPULATION
     *
     * IT CONTAINS THE METHODS THAT ARE RELATED WITH FITNESS EVALUATION AND REPRESENTATION
     *
     */

    //This method is used for evaluating the fitness score of an individual in genetic algorithm population
    public static double evaluateIndividualFitness(Problem problem, Individual iv) {

        ArrayList<Customer> customers = problem.getCustomers();

        double fitness = 0;
        for(int j = 0; j<customers.size(); j++) {

            Customer customer = customers.get(j);
            for(int k = 0; k<iv.getChromosome().size(); k++) {

                Base base = iv.getChromosome().get(k);

                //calculating the distance of a customer and checking if he/she is covered by a base
                double deltaXPow = Math.pow(customer.getX() - base.getX(), 2);
                double deltaYPow = Math.pow(customer.getY() - base.getY(), 2);
                double sqrtSum = Math.sqrt((deltaXPow + deltaYPow));


                //increate the fitness score if he/she is covered
                if(sqrtSum <= problem.getCoverRadius()){
                    fitness = fitness + 1;
                    break;
                }else {
                    continue;
                }
            }
        }

        //set the fitness level of the individual
        iv.setFitness(fitness);

        return fitness;
    }


    //This method is used for evaluating the fitness level of a whole genetic algorithm population. It is a wrapper function that uses the individual evaluation met
    public static ArrayList<Individual> evaluatePopulationFitness(Problem problem, ArrayList<Individual> population){

        for(int i = 0; i<population.size(); i++) {

            Individual iv = population.get(i);


            //evaluate and get the fitness level of the individual and set it
            double fitness = evaluateIndividualFitness(problem, iv);

            iv.setFitness(fitness);
        }

        return population;
    }







    //This method is used for showing the fitness scores of the individuals in a population one by one
    public static void showPopulationFitnessScores(Problem problem, ArrayList<Individual> population) {

        System.out.println();
        System.out.println("SHOWING POPULATION FITNESS SCORES ...");
        System.out.println();
        for(int i = 0; i<population.size(); i++) {
            double prc = (population.get(i).getFitness()/problem.getCity().getCustomerAmount())*100;
            System.out.println("INDIVIDUAL "+i+" FITNESS SCORE : " + population.get(i).getFitness() + " / "+ problem.getCity().getCustomerAmount() + " ( %" + prc + " )
        }
        System.out.println();
    }


    //This method is used for getting the average population score in a population
    public static double getAveragePopulationFitness(ArrayList<Individual> population) {

        double summation = 0;
        for(int i = 0; i<population.size(); i++) {

            summation = summation + population.get(i).getFitness();
        }

        double average = (double) (summation/population.size());

        return average;
    }


    //This method is used for printing the average population score in a genetic algorithm population
    public static void showAveragePopulationFitness(Problem problem, ArrayList<Individual> population) {

        double summation = 0;
        for(int i = 0; i<population.size(); i++) {

            summation = summation + population.get(i).getFitness();
        }

        double average = summation/population.size();

        double prc = (average/problem.getCity().getCustomerAmount())*100;
        System.out.println("AVERAGE FITNESS OF POPULATION : "+average+" / "+problem.getCity().getCustomerAmount() + " ( %" + prc + " )");
    }
}
```

```java
public class Individual implements Comparable<Individual> {

    /*
     *
     * THIS CLASS IS USED FOR DEFINING THE INDIVIDUAL OBJECT, WHICH IS THE MAIN POPULATION ACTOR IN THE GENETIC ALGORITHM
     *
     * IT CONTAINS A PROBLEM OBJECT, WHICH IS DEFINING THE PROBLEM TO BE SOLVED, A FITNESS VARIABLE WHICH DEFINES FITNESS LEVEL,
     * AND A CHROMOSOME THAT CONTAINS THE BASE COORDINATES THE INDIVIDUAL POSSESS.
     *
     */


    private int id;
    private Problem problem;
    private ArrayList<Base> chromosome;
    private Double fitness;

    private static int counter = 0;

    public Individual(Problem problem) {
        counter++;
        this.id = counter;
        this.problem = problem;
        this.chromosome = randomizeChromosome();
        this.fitness = 0.0;
    }


    //This method is used for randomizing the chromosome of the individual on creation
    public ArrayList<Base> randomizeChromosome(){

        Random r = new Random();
        double x;
        double y;
        ArrayList<Base> bases = new ArrayList<Base>();
        if(problem.getCity().getBaseLocationAmount() == 0) {

            for(int i = 0; i<problem.getCity().getBaseAmount(); i++) {

                //create random coordinates in continuous space if there are no base locations
                x = problem.getCity().getWidth() * r.nextDouble();
                y = problem.getCity().getHeight() * r.nextDouble();
                Base base = new Base(x, y, problem.getCoverRadius());
                bases.add(base);
            }

            return bases;
        }
        else {

            for(BaseLocation location  : problem.getBaseLocations()) {
                location.setOccupied(false);
            }

            int choice;
            for(int i = 0; i<problem.getCity().getBaseAmount(); i++) {

                choice = r.nextInt(problem.getBaseLocations().size());
                if(problem.getBaseLocations().get(choice).isOccupied() == true) {
                    i = i -1;
                    continue;
                }
                else {

                    //select from baselocations if they are defined.
                    x = problem.getBaseLocations().get(choice).getX();
                    y = problem.getBaseLocations().get(choice).getY();
                    Base base = new Base(x, y, problem.getCoverRadius());
```

```
            return bases;
        }
    }


    //This method is used for showing the chromosome values of the individual
    public void showChromosome() {
        System.out.println();
        System.out.println("\n INDIVIDUAL BASE COORDINATES : ");
        for(int i = 0; i<chromosome.size(); i++) {
            Base base = chromosome.get(i);
            System.out.print("Base "+i+" ("+
                             String.format("%.2f", base.getX())+
                             " / "+
                             String.format("%.2f", base.getY())+
                             ") , ");
        }
        System.out.println();
    }

    public static int getIndividualAmount() {
        return counter;
    }

    public Problem getProblem() {
        return problem;
    }

    public void setProblem(Problem problem) {
        this.problem = problem;
    }

    public ArrayList<Base> getChromosome() {
        return chromosome;
    }

    public void setChromosome(ArrayList<Base> chromosome) {
        this.chromosome = chromosome;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public Double getFitness() {
        return fitness;
    }

    public void setFitness(Double fitness) {
        this.fitness = fitness;
    }

    @Override
    public int compareTo(Individual o) {

        if (getFitness() == 0 || o.getFitness() == 0) {
            return 0;
        }

        return getFitness().compareTo(o.getFitness());
    }

}
```

```java
public class Mating {

    /*
     *
     * THIS CLASS IS FOR DEFINING THE METHODS THAT ARE USED FOR GENETIC ALGORITHM INDIVIDUAL AND POPULATION BREEDING FUNCTIONS
     *
     */


    //This method is used for breeding a population and applying crossover in a given rate. It is actually a wrapper function that uses individual breeding function.
    public static ArrayList<Individual> breedPopulation(Problem problem, ArrayList<Individual> population, double amountChildren, double crossoverRate) {

        Random r = new Random();
        ArrayList<Individual> children = new ArrayList<Individual>();
        Individual child = new Individual(problem);

        for(int i = 0; i<amountChildren; i++) {

            //select parents from the population
            int parent1Index = r.nextInt(population.size());
            int parent2Index = r.nextInt(population.size());

            Individual parent1 = population.get(parent1Index);
            Individual parent2 = population.get(parent2Index);

            //produce the children
            child = produceChildrenByCrossover(problem, parent1, parent2, crossoverRate);
            children.add(child);
        }

        return children;
    }


    //This method is used for producing children from parent individuals and applies crossover in the given rate.
    public static Individual produceChildrenByCrossover(Problem problem, Individual parent1, Individual parent2, double crossoverRate) {

        Random r = new Random();
        int coDie = r.nextInt(100);
        Individual child = new Individual(problem);
        ArrayList<Base> chromosome = new ArrayList<Base>();
        crossoverRate = crossoverRate*100;

        //apply crossover if the die is lower than the rate
        if(coDie < crossoverRate) {

            int coIndex = r.nextInt(parent1.getChromosome().size());

            //split and merge the chromosomes of the parents
            for(int i = 0; i<coIndex; i++) {
                chromosome.add(parent1.getChromosome().get(i));
            }

            for(int i =coIndex; i<parent1.getChromosome().size(); i++) {
                chromosome.add(parent2.getChromosome().get(i));
            }

            child.setChromosome(chromosome);

        }
        else {
            int choice = r.nextInt(2);

            //dont apply crossover and select a random parents chromosome
            if(choice == 1) {
```

```java
    //This method is used for producing children from parent individuals and applies crossover in the given rate.
    public static Individual produceChildrenByCrossover(Problem problem, Individual parent1, Individual parent2, double crossoverRate) {

        Random r = new Random();
        int coDie = r.nextInt(100);
        Individual child = new Individual(problem);
        ArrayList<Base> chromosome = new ArrayList<Base>();
        crossoverRate = crossoverRate*100;

        //apply crossover if the die is lower than the rate
        if(coDie < crossoverRate) {

            int coIndex = r.nextInt(parent1.getChromosome().size());

            //split and merge the chromosomes of the parents
            for(int i = 0; i<coIndex; i++) {
                chromosome.add(parent1.getChromosome().get(i));
            }

            for(int i =coIndex; i<parent1.getChromosome().size(); i++) {
                chromosome.add(parent2.getChromosome().get(i));
            }

            child.setChromosome(chromosome);

        }
        else {
            int choice = r.nextInt(2);

            //dont apply crossover and select a random parents chromosome
            if(choice == 1) {
                chromosome = parent1.getChromosome();
            }
            else {
                chromosome = parent2.getChromosome();
            }

            child.setChromosome(chromosome);

        }

        return child;
    }

}
```

```java
public class Mutation {

    /*
     *
     * THIS CLASS IS USED FOR DEFINING THE METHODS FOR APPLYING MUTATION TO THE GENETIC ALGORITHM INDIVIDALS AND POPULATION
     *
     */


    //this method is used for mutating the whole genetic algorithm population and acts as a wrapper function that reaches the individual mutation function
    public static ArrayList<Individual> mutatePopulation(Problem problem, ArrayList<Individual> population, double mutationRate) {

        ArrayList<Individual> mutatedPopulation = new ArrayList<Individual>();
        for (int i = 0; i<population.size(); i++) {

            //mutate each individual
            Individual mutatedIv = mutateIndividual(problem, population.get(i), mutationRate);
            mutatedPopulation.add(mutatedIv);
        }

        return mutatedPopulation;
    }


    //this method is used for mutating a single individual with a given mutation rate.
    public static Individual mutateIndividual(Problem problem, Individual iv, double mutationRate) {

        Random r = new Random();
        int muteDie = r.nextInt(100);
        Individual mutatedIv = new Individual(problem);
        mutationRate = mutationRate*100;

        //if the mutation die is smaller than the mutation rate, apply mutation; otherwise leave as it is.
        if(muteDie < mutationRate) {

            int mutationIndex = r.nextInt(iv.getChromosome().size());
            ArrayList<Base> mutatedChromosome = mutateGene(problem, iv.getChromosome(), mutationIndex);
            mutatedIv.setChromosome(mutatedChromosome);

        }
        else {

            mutatedIv = iv;

        }

        return mutatedIv;
    }


    //this method is used for mutating a single gene in a chromosome.
    public static ArrayList<Base> mutateGene(Problem problem, ArrayList<Base> chromosome, int mutationIndex) {

        Random r = new Random();
        double x;
        double y;
        double coverRadius = problem.getCoverRadius();
        Base base;

        //apply mutation according to the problem structure.
        if(problem.getCity().getBaseLocationAmount() == 0) {

            x = problem.getCity().getWidth() * r.nextDouble();
            y = problem.getCity().getHeight() * r.nextDouble();
            base = new Base(x, y, coverRadius);
        }
        else {
```

```java
    //this method is used for mutating a single gene in a chromosome.
    public static ArrayList<Base> mutateGene(Problem problem, ArrayList<Base> chromosome, int mutationIndex) {

        Random r = new Random();
        double x;
        double y;
        double coverRadius = problem.getCoverRadius();
        Base base;

        //apply mutation according to the problem structure.
        if(problem.getCity().getBaseLocationAmount() == 0) {

            x = problem.getCity().getWidth() * r.nextDouble();
            y = problem.getCity().getHeight() * r.nextDouble();
            base = new Base(x, y, coverRadius);
        }
        else {

            ArrayList<BaseLocation> baseLocations = problem.getBaseLocations();
            int baseChoice = r.nextInt(baseLocations.size());
            BaseLocation baseLoc = baseLocations.get(baseChoice);

            x = baseLoc.getX();
            y = baseLoc.getY();
            base = new Base(x, y, coverRadius);
        }

        chromosome.set(mutationIndex, base);
        return chromosome;
    }
}
```

```java
public class Population {


    /*
     *
     * THIS CLASS IS USED FOR GENERATING POPULATIONS FOR GENETIC ALGORITHMS WITH SPECIFIED PARAMETERS
     *
     */


    //This method is used for generating a population for the genetic algorithm
    public static ArrayList<Individual> generatePopulation(Problem problem, int populationSize){

        ArrayList<Individual> population = new ArrayList<Individual>();
        for(int i = 0; i<populationSize; i++) {
            Individual individual = new Individual(problem);
            population.add(individual);
        }

        return population;
    }


    //This method is used for showing the basic data about the population including fitness scores.
    public static void showPopulationData(ArrayList<Individual> population) {

        System.out.println();
        System.out.println("PRINTING THE INFORMATION OF THE POPULATION ... ");
        System.out.println();
        for(int i = 0; i<population.size(); i++) {
            Individual iv = population.get(i);
            System.out.println("INDIVIDUAL NUMBER: " + i);
            System.out.println("INDIVIDUAL FITNESS SCORE: " + iv.getFitness());
            iv.showChromosome();
            System.out.println("_________________");
        }
    }


    //This method is used for combining the different population groups in the ga; such as the elites, elders, children and random individuals
    public static ArrayList<Individual> combinePopulation(ArrayList<Individual> elders, ArrayList<Individual> children, ArrayList<Individual> randoms){

        ArrayList<Individual> combinedPopulation = new ArrayList<Individual>();

        for(int i = 0; i<elders.size(); i++) {
            combinedPopulation.add(elders.get(i));
        }

        for(int i = 0; i<children.size(); i++) {
            combinedPopulation.add(children.get(i));
        }

        for(int i = 0; i<randoms.size(); i++) {
            combinedPopulation.add(randoms.get(i));
        }

        //return the combined population
        return combinedPopulation;
    }
}
```

```java
public class Selection {


    /*
     *
     * THIS CLASS IS USED FOR DEFINING THE SELECTION METHODS FOR THE GENETIC ALGORITHM
     *
     */



    //this method is used for selecting elite individuals from the population according to their fitness level.
    public static ArrayList<Individual> selectElitesByFitness(ArrayList<Individual> population,
                                                              double elitesAmount){

        //define the elites group and the sorted population (by fitness score)
        ArrayList<Individual> elites = new ArrayList<Individual>();
        ArrayList<Individual> sortedPopulation = sortPopulationByFitness(population);

        for(int i = 0; i<elitesAmount; i++) {

            elites.add(sortedPopulation.get(i));
        }

        return elites;
    }


    //this method is used for selecting successful elders that will pass to other generation by their fitness score.
    public static ArrayList<Individual> selectElders(ArrayList<Individual> population,
                                ArrayList<Individual> elites,
                                double eldersAmount) {

        ArrayList<Individual> elders = new ArrayList<Individual>();

        for(int i = 0; i<eldersAmount; i++) {

            elders.add(elites.get(i));
        }

        return elders;
    }


    //this method is used for randomly selecting individuals from the population to pass to next generation
    public static ArrayList<Individual> randomlySelectIndividuals(ArrayList<Individual> population,
                                                          double randomAmount){

        ArrayList<Individual> randomIvs = new ArrayList<Individual>();

        Random r = new Random();
        for(int i = 0; i<randomAmount; i++) {

            int choice = r.nextInt(population.size());
            randomIvs.add(population.get(choice));
        }

        return randomIvs;
    }


    //this method is used for sorting the population arraylist by fitness score.
    public static ArrayList<Individual> sortPopulationByFitness(ArrayList<Individual> population){

        Collections.sort(population);
        Collections.reverse(population);

        return population;
    }
```

(MODEL PAKETİ)

```java
public class Base {

    /*
     *
     * THIS CLASS DEFINES THE BASE OBJECT THAT WILL BE USED FOR DESCRIBING THE GENE IN THE CHROMOSOME OF THE INDIVIDUAL
     *
     * EACH BASE HAS A LOCATION IN THE 2-DIMENSIONAL SPACE SUCH THAT X -> HORIZONTAL  &  Y -> VERTICAL AXIS.
     *
     * EACH BASE ALSO HAS A COVER RADIUS WHICH MEANS HOW FAR THEY CAN SERVE THE TELECOMMUNICATION SERVICES AT MAXIMUM.
     *
     */

    private int id;
    private double x;
    private double y;
    private double coverRadius;


    private static int counter = 0;


    public Base(double x, double y, double coverRadius) {
        counter ++;
        this.id = counter;
        this.x = x;
        this.y = y;
        this.coverRadius = coverRadius;
    }

    public static int getBaseAmount() {
        return counter;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public double getX() {
        return x;
    }

    public void setX(double x) {
        this.x = x;
    }

    public double getY() {
        return y;
    }

    public void setY(double y) {
        this.y = y;
    }

    public double getCoverRadius() {
        return coverRadius;
    }

    public void setCoverRadius(double coverRadius) {
        this.coverRadius = coverRadius;
    }

}
```

```java
public class BaseLocation {

    /*
     *
     * THIS CLASS DEFINES THE BASE LOCATION OBJECT WHICH IS REPRESENTING THE POSSIBLE LOCATIONS INSIDE A CITY
     * THAT THE TELECOMMUNICATIONS COMPANY CAN USE TO PUT THE BASE OBJECTS.
     *
     * EACH BASELOCATION HAS 2-DIMENSIONAL COORDINATES IN THE CITY SUCH THAT X -> HORIZONTAL & Y -> VERTICAL AXIS.
     *
     * EACH BASELOCATION ALSO HAS AN ASPECT CALLED isOccupied, SO WHILE RANDOMIZING THE BASES, THERE CAN'T BE MORE THAN 1 BASES AT THE SAME BASE LOCATIONS
     *
     */

    private int id;
    private double x;
    private double y;
    private boolean isOccupied;

    private static int counter = 0;

    public BaseLocation(double x, double y) {
        counter++;
        this.id = counter;
        this.x = x;
        this.y = y;
        this.isOccupied = false;
    }

    public static int getBaseLocationAmount() {
        return counter;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public double getX() {
        return x;
    }

    public void setX(double x) {
        this.x = x;
    }

    public double getY() {
        return y;
    }

    public void setY(double y) {
        this.y = y;
    }

    public boolean isOccupied() {
        return isOccupied;
    }

    public void setOccupied(boolean isOccupied) {
        this.isOccupied = isOccupied;
    }

}
```

```java
public class City {

    /*
     *
     * CITY IS THE OBJECT, REPRESENTING THE 2-DIMENSIONAL SPACE THAT THE TELECOMMUNICATIONS COMPANY AIMS TO WORK ON TO PUT THE BASE OBJECTS
     *
     * CITIES HAVE A WIDTH, REPRESENTING THE LARGENESS ON HORIZONTAL AXIS, AND A HEIGHT, REPRESENTING THE LARGENESS ON VERTICAL AXIS.
     *
     * CITIES CAN HAVE SPECIFIC BASE LOCATIONS OR NOT. THAT DEPENDS ON THE CHOICE OF THE USER AND IF THE CITY HAS SPECIFIC BASE LOCATIONS,
     * THAT MEANS THAT THE BASES CAN ONLY BE PUT THERE. BUT IF THERE ISN'T, ONE CAN PUT THE BASES ANYWHERE IN THE CITY.
     *
     * CITIES ALSO HAVE SPECIFIC CUSTOMER AMOUNTS AND BASE AMOUNTS THEY COULD CARRY, WHICH IS AGAIN TO THE CHOICE OF THE USER.
     *
     */

    private int id;
    private double width;
    private double height;
    private int baseLocationAmount;
    private int customerAmount;
    private int baseAmount;

    private static int counter = 0;

    public City(double width, double height, int customerAmount, int baseLocationAmount, int baseAmount) {
        counter++;
        this.id = counter;
        this.width = width;
        this.height = height;
        this.customerAmount = customerAmount;
        this.baseLocationAmount = baseLocationAmount;
        this.baseAmount = baseAmount;
    }

    public City(double width, double height, int customerAmount, int baseAmount) {
        counter++;
        this.id = counter;
        this.width = width;
        this.height = height;
        this.customerAmount = customerAmount;
        this.baseLocationAmount = 0;
        this.baseAmount = baseAmount;
    }

    public static int getCityAmount() {
        return counter;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public double getWidth() {
        return width;
    }

    public void setWidth(double width) {
        this.width = width;
    }
```

```java
public class Customer {

    /*
     *
     * CUSTOMERS ARE REPRESENTING THE MAJOR FACTOR THAT AFFECTS THE FITNESS AS THE AIM IS TO OPTIMIZE THE NUMBER OF CUSTOMERS
     * WHO CAN ACHIEVE SIGNALS FROM THE TELECOMMUNICATION BASES.
     *
     * EACH CUSTOMER HAS SPECIFIC COORDINATES IN THE CITY / 2-DIMENSIONAL SPACE SUCH THAT X -> HORIZONTAL AXIS & Y -> VERTICAL AXIS
     *
     *
     */

    private int id;
    private double x;
    private double y;

    private static int counter = 0;

    public Customer(double x, double y) {
        counter++;
        this.id = counter;
        this.x = x;
        this.y = y;
    }

    public static int getCustomerAmount() {
        return counter;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public double getX() {
        return x;
    }

    public void setX(double x) {
        this.x = x;
    }

    public double getY() {
        return y;
    }

    public void setY(double y) {
        this.y = y;
    }

}
```

```java
public class Problem {


    /*
     *
     * PROBLEM IS THE OBJECT THAT ACTS AS A WRAPPER OF THE PROBLEM DEFINITION FOR OUR OPTIMIZATION PROBLEMS
     * IT CAN WORK WITH DIFFERENT RANDOMIZED SETS OF PARAMETERS AND IT CAN ALSO WORK ON DIFFERENT CITIES WITH DIFFERENT PARAMETERS,
     * DIFFERENT COVER RADIUS -> WHICH IS THE MAXIMUM DISTANCE A BASE CAN SERVE.
     *
     */

    private City city;
    private ArrayList<Customer> customers;
    private ArrayList<BaseLocation> baseLocations;
    private double coverRadius;


    public Problem(City city, double coverRadius) {
        this.city = city;
        this.customers = randomizeCustomers();
        System.out.println("\n PROBLEM - CUSTOMERS AND BASE LOCATIONS : ");

        //Show customer coordinates on problem creation
        showCustomerCoordinates();
        this.baseLocations = randomizeBaseLocations();

        //Show base location coordinates on problem creation (if exists)
        showBaseLocationCoordinates();
        this.coverRadius = coverRadius;
    }


    //This method randomizes customer objects and their locations on the specified city.
    public ArrayList<Customer> randomizeCustomers(){

        Random r = new Random();
        ArrayList<Customer> customers = new ArrayList<Customer>();
        double x;
        double y;
        for(int i =0; i<city.getCustomerAmount(); i++) {
            x = city.getWidth() * r.nextDouble();
            y = city.getHeight() * r.nextDouble();
            Customer customer = new Customer(x, y);
            customers.add(customer);
        }

        return customers;
    }
```

```java
    //this method randomizes base location objects and their locations on the specified city.
    public ArrayList<BaseLocation> randomizeBaseLocations(){

        if(city.getBaseLocationAmount() == 0) {
            return null;
        }
        else {

            Random r = new Random();
            ArrayList<BaseLocation> baseLocations = new ArrayList<BaseLocation>();
            double x;
            double y;
            for(int i = 0; i<city.getBaseLocationAmount(); i++) {
                x = city.getWidth() * r.nextDouble();
                y = city.getHeight() * r.nextDouble();
                BaseLocation baseLocation = new BaseLocation(x, y);
                baseLocations.add(baseLocation);
            }

            return baseLocations;
        }

    }

    public City getCity() {
        return city;
    }

    public void setCity(City city) {
        this.city = city;
    }

    public ArrayList<BaseLocation> getBaseLocations() {
        return baseLocations;
    }

    public void setBaseLocations(ArrayList<BaseLocation> baseLocations) {
        this.baseLocations = baseLocations;
    }

    public ArrayList<Customer> getCustomers() {
        return customers;
    }

    public void setCustomers(ArrayList<Customer> customers) {
        this.customers = customers;
    }

    public double getCoverRadius() {
        return coverRadius;
    }

    public void setCoverRadius(double coverRadius) {
        this.coverRadius = coverRadius;
    }


    //This method makes us be able to see the coordinates of the randomized customers on creation
    public void showCustomerCoordinates() {

        System.out.println("\n CUSTOMER LOCATIONS (X / Y) : ");
        for(int i = 0; i<this.customers.size(); i++) {
            System.out.println("Customer "+i+" ("+
                               String.format("%.2f", customers.get(i).getX())+
                               " / "+
                               String.format("%.2f", customers.get(i).getY())+
                               ") , ");
```

```java
    //This method makes us able to see the coordinates of the randomized base locations on creation (if exists)
    public void showBaseLocationCoordinates() {

        if(city.getBaseLocationAmount() != 0) {
            System.out.println("\n BASE LOCATION COORDINATES (X / Y) : ");
            for(int i = 0; i<baseLocations.size(); i++) {
                System.out.println("Base Location "+i+" ("+
                        String.format("%.2f", baseLocations.get(i).getX())+
                        " / "+
                        String.format("%.2f", baseLocations.get(i).getY())+
                        ") , ");
            }
            System.out.println();
        }
        else {

            //If there is no base location, inform the user that the system is continuous.
            System.out.println("\n This city does not have mandatory base locations! The problem is continuous!");
            System.out.println();
        }
    }

}
```

(OPTİMİZASYON YARATICI MAKRO PAKETİ)

```java
public class OptimizationBuilder {

    /*
     *
     * OPTIMIZATION BUILDER IS A CLASS THAT CONTAINS WRAPPER FUNCTIONS THAT RUNS THE FOLLOWING OPTIMIZATION ALGORITHMS:
     *      -> GENETIC ALGORITHM
     *      -> CONTINUOUS OPTIMIZATION -> DIFFERENTIAL EVOLUTION IS USED
     *              ->>> DE/RAND/1
     *              ->>> DE/RAND/2
     *              ->>> DE/BEST/1
     *              ->>> DE/BEST/2
     *              ->>> DE/CURRENT_TO_BEST/1
     *
     *
     */


    //This method is used for iteratively pushing the generations in the genetic algorithm further. Defined as a separate function to increase modularity.
    public static ArrayList<Individual> nextGenerationGA(Problem problem, ArrayList<Individual> population,
                                                         double elitismAmount, double elderAmount,
                                                         double randomIndividualAmount, double childrenAmount,
                                                         double crossoverRate, double mutationRate) {

        //evaluate population fitness
        population = Fitness.evaluatePopulationFitness(problem, population);

        //show average population fitness
        Fitness.showAveragePopulationFitness(problem, population);

        //select best individuals/elites from the population
        ArrayList<Individual> elites = Selection.selectElitesByFitness(population, elitismAmount);

        //select elders from elites
        ArrayList<Individual> elders = Selection.selectElders(population, elites, elderAmount);

        //select random individuals from the population
        ArrayList<Individual> randomIvs = Selection.randomlySelectIndividuals(population, randomIndividualAmount);

        //produce children from the elite individuals
        ArrayList<Individual> children = Mating.breedPopulation(problem, elites, childrenAmount, crossoverRate);

        //combine the population
        ArrayList<Individual> combinedPopulation = Population.combinePopulation(elders, children, randomIvs);

        //mutate the children
        children = Mutation.mutatePopulation(problem, combinedPopulation, mutationRate);

        //return the resulting population (children)
        return children;
    }




    //This method is used for iteratively pushing the generations in the differential evolution (DE/rand/1) further. Defined as a separate function to increase modu
    public static ArrayList<DEIndividual> nextGenerationContinuousOptimization_DErand1(Problem problem, ArrayList<DEIndividual> population,
                                                                                        double F, double crossoverRate) {

        //evaluate the fitness of the population
        population = DEFitness.evaluateDEPopulationFitness(problem, population);

        //show the average fitness of the population
        DEFitness.showAverageDEPopulationFitness(problem, population);

        //mutate the individuals with mutation method de/rand/1
        ArrayList<DEIndividual> mutatedIndividuals = DECrossover.mutateDEPopulation_DErand1(problem, population, F, crossoverRate);

        //return mutated individuals
        return mutatedIndividuals;
    }

    //This method is used for iteratively pushing the generations in the differential evolution (DE/rand/2) further. Defined as a separate function to increase modu
    public static ArrayList<DEIndividual> nextGenerationContinuousOptimization_DErand2(Problem problem, ArrayList<DEIndividual> population,
                                                                                        double F, double crossoverRate) {

        //evaluate the fitness of the population
        population = DEFitness.evaluateDEPopulationFitness(problem, population);

        //show the average fitness of the population
        DEFitness.showAverageDEPopulationFitness(problem, population);

        //mutate the individuals with mutation method de/rand/2
        ArrayList<DEIndividual> mutatedIndividuals = DECrossover.mutateDEPopulation_DErand2(problem, population, F, crossoverRate);

        //return the mutated individuals
        return mutatedIndividuals;
    }

    //This method is used for iteratively pushing the generations in the differential evolution (DE/best/1) further. Defined as a separate function to increase modu
    public static ArrayList<DEIndividual> nextGenerationContinuousOptimization_DEbest1(Problem problem, ArrayList<DEIndividual> population,
                                                                                        double F, double crossoverRate) {

        //evalute the fitness of the population
        population = DEFitness.evaluateDEPopulationFitness(problem, population);

        //show the average fitness of the population
        DEFitness.showAverageDEPopulationFitness(problem, population);

        //mutate the individuals with mutation method de/best/1
        ArrayList<DEIndividual> mutatedIndividuals = DECrossover.mutateDEPopulation_DEbest1(problem, population, F, crossoverRate);

        //return the mutated individuals
        return mutatedIndividuals;
    }
```

```java
//This method is used for iteratively pushing the generations in the differential evolution (DE/current_to_best/1) further. Defined as a separate function to
public static ArrayList<DEIndividual> nextGenerationContinuousOptimization_DEcurToBest1(Problem problem, ArrayList<DEIndividual> population,
                                                                                         double F, double crossoverRate) {

    //evaluate the fitness of the population
    population = DEFitness.evaluateDEPopulationFitness(problem, population);

    //show the average fitness of the population
    DEFitness.showAverageDEPopulationFitness(problem, population);

    //mutate the individuals with mutation method de/current_to_best/1
    ArrayList<DEIndividual> mutatedIndividuals = DECrossover.mutateDEPopulation_DEcurToBest1(problem, population, F, crossoverRate);

    //return mutated individuals
    return mutatedIndividuals;
}


//This is the main wrapper function to be able to use for running the genetic algorithm with predefined hyper-parameters
public static ArrayList<Individual> runGA(Problem problem, int populationSize, int maxGenerations,
                            double elitismAmount, double elderAmount,
                            double randomIndividualAmount, double childrenAmount,
                            double crossoverRate, double mutationRate) {

    //generate initial population
    ArrayList<Individual> population = Population.generatePopulation(problem, populationSize);

    //evaluate initial population fitness
    Fitness.evaluatePopulationFitness(problem, population);

    //show initial average population fitness
    Fitness.showAveragePopulationFitness(problem, population);

    //Run the algorithm for maxGenerations times
    System.out.println("GENERATION 0 : \n");

    for(int i = 0; i<maxGenerations; i++) {

        System.out.println("GENERATION :" + i + " \n");

        ArrayList<Individual> nextPopulation = nextGenerationGA(problem, population, elitismAmount, elderAmount,
                                                                randomIndividualAmount, childrenAmount,
                                                                crossoverRate, mutationRate);

        System.out.println("\n_____");

        population = nextPopulation;
    }

    //evaluate final populationn fitness
    population = Fitness.evaluatePopulationFitness(problem, population);

    //return final population
    return population;

}
```

```java
//This is the main wrapper function to be able to use for running the differential evolution (DE/rand/1) with predefined hyper-parameters
public static ArrayList<DEIndividual> runContinuousOptimization_DErand1(Problem problem, int populationSize, int maxGenerations,
                                                                         double F, double crossoverRate) {

    //Generate initial population
    ArrayList<DEIndividual> population = DEPopulation.generateDEPopulation(problem, populationSize);

    //evaluate initial population fitness
    DEFitness.evaluateDEPopulationFitness(problem, population);

    //show average initial population fitness
    DEFitness.showAverageDEPopulationFitness(problem, population);

    //run the algorithm for maxGenerations times
    System.out.println("GENERATION 0 : \n");

    for(int i = 0; i<maxGenerations; i++) {

        System.out.println("GENERATION :" + i + " \n");

        ArrayList<DEIndividual> nextPopulation = nextGenerationContinuousOptimization_DErand1(problem, population, F, crossoverRate);

        System.out.println("\n_____");

        population = nextPopulation;
    }

    //Evaluate final population fitness
    population = DEFitness.evaluateDEPopulationFitness(problem, population);

    //return final population
    return population;
}


//This is the main wrapper function to be able to use for running the differential evolution (DE/rand/2) with predefined hyper-parameters
public static ArrayList<DEIndividual> runContinuousOptimization_DErand2(Problem problem, int populationSize, int maxGenerations,
                                                                         double F, double crossoverRate) {

    //generate initial population
    ArrayList<DEIndividual> population = DEPopulation.generateDEPopulation(problem, populationSize);

    //evaluate initial population fitness
    DEFitness.evaluateDEPopulationFitness(problem, population);

    //show average initial population fitness
    DEFitness.showAverageDEPopulationFitness(problem, population);

    //run the algorithm for maxGenerations times
    System.out.println("GENERATION 0 : \n");

    for(int i = 0; i<maxGenerations; i++) {

    System.out.println("GENERATION :" + i + " \n");

    ArrayList<DEIndividual> nextPopulation = nextGenerationContinuousOptimization_DErand2(problem, population, F, crossoverRate);

    System.out.println("\n_____");

    population = nextPopulation;
    }

    //evaluate the final population fitness
    population = DEFitness.evaluateDEPopulationFitness(problem, population);

    //return final population
    return population;
}
```

```java
    //This is the main wrapper function to be able to use for running the differential evolution (DE/best/1) with predefined hyper-parameters
    public static ArrayList<DEIndividual> runContinuousOptimization_DEbest1(Problem problem, int populationSize, int maxGenerations,
            double F, double crossoverRate) {


        //generate initial population
        ArrayList<DEIndividual> population = DEPopulation.generateDEPopulation(problem, populationSize);

        //evaluate initial population fitness
        DEFitness.evaluateDEPopulationFitness(problem, population);

        //show initial population fitness
        DEFitness.showAverageDEPopulationFitness(problem, population);


        //Run the algorithm for maxGenerations times
        System.out.println("GENERATION 0 : \n");

        for(int i = 0; i<maxGenerations; i++) {

        System.out.println("GENERATION :" + i + " \n");

        ArrayList<DEIndividual> nextPopulation = nextGenerationContinuousOptimization_DEbest1(problem, population, F, crossoverRate);

        System.out.println("\n_____");

        population = nextPopulation;
        }

        //evaluate the fitness for the final population
        population = DEFitness.evaluateDEPopulationFitness(problem, population);

        //return the final population
        return population;
    }


    //This is the main wrapper function to be able to use for running the differential evolution (DE/best/2) with predefined hyper-parameters
    public static ArrayList<DEIndividual> runContinuousOptimization_DEbest2(Problem problem, int populationSize, int maxGenerations,
            double F, double crossoverRate) {

        //generate initial population
        ArrayList<DEIndividual> population = DEPopulation.generateDEPopulation(problem, populationSize);

        //evaluate initial population fitness
        DEFitness.evaluateDEPopulationFitness(problem, population);

        //show average initial population fitness
        DEFitness.showAverageDEPopulationFitness(problem, population);


        //Run the algorithm for maxGenerations times.
        System.out.println("GENERATION 0 : \n");

        for(int i = 0; i<maxGenerations; i++) {

        System.out.println("GENERATION :" + i + " \n");

        ArrayList<DEIndividual> nextPopulation = nextGenerationContinuousOptimization_DEbest2(problem, population, F, crossoverRate);

        System.out.println("\n_____");

        population = nextPopulation;
        }

        //evaluate the final population
        population = DEFitness.evaluateDEPopulationFitness(problem, population);
```

```java
    //This is the main wrapper function to be able to use for running the differential evolution (DE/current_to_best/1) with predefined hyper-parameters
    public static ArrayList<DEIndividual> runContinuousOptimization_DEcurToBest1(Problem problem, int populationSize, int maxGenerations,
            double F, double crossoverRate) {


        //Generate initial population
        ArrayList<DEIndividual> population = DEPopulation.generateDEPopulation(problem, populationSize);

        //Evaluate the initial population fitness
        DEFitness.evaluateDEPopulationFitness(problem, population);

        //Show the average population fitness
        DEFitness.showAverageDEPopulationFitness(problem, population);


        //Run the algorithm for maxGenerations times
        System.out.println("GENERATION 0 : \n");

        for(int i = 0; i<maxGenerations; i++) {

        System.out.println("GENERATION :" + i + " \n");

        ArrayList<DEIndividual> nextPopulation = nextGenerationContinuousOptimization_DEcurToBest1(problem, population, F, crossoverRate);

        System.out.println("\n_____");

        population = nextPopulation;
        }

        //evaluate the fitness of the final population
        population = DEFitness.evaluateDEPopulationFitness(problem, population);

        //return the final population
        return population;
    }

}
```

(YARDIMCI PAKET)

```java
public class MathSupport {

    /*
     *  THIS CLASS CONTAINS THE MATHEMATICAL SUPPORT FUNCTIONS THAT THE PROGRAM USES FOR EVALUATING METRICS FOR SOLUTIONS.
     *
     */


    //This method calculates the standard deviation of the given elements in an arraylist
    public static double sd (ArrayList<Double> results)
    {

        double average = average(results);
        double temp = 0;

        for (int i = 0; i < results.size(); i++)
        {
            double val = results.get(i);

            double squrDiffToMean = Math.pow(val - average, 2);

            temp += squrDiffToMean;
        }

        double meanOfDiffs = (double) temp / (double) (results.size());

        return Math.sqrt(meanOfDiffs);
    }


    //This method calculates the median of the given elements in an arraylist
    public static double median (ArrayList<Double> results) {

        double median = (results.get(results.size()/2) + results.get(results.size()/2 - 1))/2;
        return median;
    }


    //This method takes the average of the given elements in an arraylist
    public static double average (ArrayList<Double> results) {

        double summation = 0;
        for (int i = 0; i<results.size(); i++) {
            summation = summation + results.get(i);
        }

        double average = summation/results.size();
        return average;
    }




    //This method takes the best result in a given arraylist and mainly used for taking the maximum fitness value in a metrics array.
    public static double getBestResult(ArrayList<Double> results){

        double max = 0;
        for(int i = 0; i<results.size(); i++) {

            if(results.get(i) > max) {
                max = results.get(i);
            }
            else {
                continue;
            }
        }

        return max;
    }


    //This method takes the worst result in a given arraylist and mainly used for taking the minimum fitness value in a metrics array.
    public static double getWorstResult(ArrayList<Double> results){

        double min = results.get(0);
        for(int i = 0; i<results.size(); i++) {

            if(results.get(i) < min) {
                min = results.get(i);
            }

        }

        return min;
    }

}
```

```java
public class PrintSupport {

    /*
     *
     *  THIS CLASS CONTAINS THE METHODS THAT IS USED FOR PRINTING THE METRICS FOR DIFFERENT SETS OF CALCULATIONS WITH DIFFERENT PARAMETERS
     *
     * */


    //This method prints the multi-metrics for an increased number of calculations with different parameter sets.
    public static void printMultiMetrics(Problem problem, ArrayList<ArrayList<Double>> multiResultsAvg, ArrayList<ArrayList<Double>> multiMetrics) {

        for(int i = 0; i<multiResultsAvg.size(); i++) {

            System.out.println("RESULTS FOR PARAMETER SET - "+i+" : ");

            ArrayList<Double> resultsAvg = multiResultsAvg.get(i);

            for(int j = 0; j<resultsAvg.size(); j++ ) {
                System.out.println("RUN "+ (j+1) + " : " + resultsAvg.get(j)  + " / "+ problem.getCity().getCustomerAmount() + " ( %" + (resultsAvg.get(j)/problem.g
            }
            System.out.println("_____________________");

            ArrayList<Double> metrics = multiMetrics.get(i);


            System.out.println("AVERAGE : " + metrics.get(0)  + " / "+ problem.getCity().getCustomerAmount() + " ( %" + (metrics.get(0)/problem.getCity().getCustome
            System.out.println("STANDARD DEVIATION : "+ metrics.get(1) + " / "+ problem.getCity().getCustomerAmount() + " ( %" + (metrics.get(1)/problem.getCity().g
            System.out.println("BEST RESULT : "+ metrics.get(2)  + " / "+ problem.getCity().getCustomerAmount() + " ( %" + (metrics.get(2)/problem.getCity().getCus
            System.out.println("WORST RESULT : " + metrics.get(3)  + " / "+ problem.getCity().getCustomerAmount() + " ( %" + (metrics.get(3)/problem.getCity().getCu
            System.out.println("MEDIAN : " + metrics.get(4)  + " / "+ problem.getCity().getCustomerAmount() + " ( %" + (metrics.get(4)/problem.getCity().getCustome

            System.out.println("**********************");
            System.out.println();


            }

    }
}
```

(SÜREKLİ OPTİMİZASYON ALGORİTMASI TEST)

```java
public class COGeneralTest {

    public static void main(String[] args) {

        /*
         *  THIS CLASS IS USED FOR TESTING THE USE OF A CONTINUOUS OPTIMIZATION ALGORITHM FOR OPTIMIZING
         *  THE PROBLEM THAT WAS MENTIONED IN "HOMEWORK PART 3 : CITY B"
         *
         *  USED METHOD : DIFFERENTIAL EVOLUTION
         *
         *  I HAVE USED 5 DIFFERENT MUTATION METHODS :
         *  1) DE/RAND/1
         *  2) DE/RAND/2
         *  3) DE/BEST/1
         *  4) DE/BEST/2
         *  5) DE/CURRENT_TO_BEST/1
         *
         *  BY USING THIS PAGE, YOU CAN TEST ALL OF THESE DIFFERENT MUTATION METHODS ON THE SAME PROBLEM.
         *
         */


        //Continuous Optimization Hyper-parameters
        int populationSize = 100;
        int maxGenerations = 2000;
        double F = 0.1;
        double crossoverRate = 0.9;

        //Multi-Solution Hyper-parameters
        double parameterRandomizationTimes = 4;
        double problemSolutionTimes = 25;

        //Achieve the problem from global variables class
        Problem problemB = GlobalDEVariables.problemB;
        problemB.showCustomerCoordinates();

        //Define the wrapper arrays that will hold the metrics from multiple calculations with multiple parameter sets
        ArrayList<ArrayList<Double>> multiMetrics1 = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiMetrics2 = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiMetrics3 = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiMetrics4 = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiMetrics5 = new ArrayList<ArrayList<Double>>();

        ArrayList<ArrayList<Double>> multiResultsAvg1 = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiResultsAvg2 = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiResultsAvg3 = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiResultsAvg4 = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiResultsAvg5 = new ArrayList<ArrayList<Double>>();




        //Run the continuous optimization algorithm
        for(int i = 0; i<parameterRandomizationTimes; i++) {

            //For each parameter set, define 5 array lists that will hold the result metrics of 5 different differential evolution mutation methods.
            ArrayList<Double> resultsAvgFitness1 = new ArrayList<Double>();
            ArrayList<Double> resultsAvgFitness2 = new ArrayList<Double>();
            ArrayList<Double> resultsAvgFitness3 = new ArrayList<Double>();
            ArrayList<Double> resultsAvgFitness4 = new ArrayList<Double>();
            ArrayList<Double> resultsAvgFitness5 = new ArrayList<Double>();

            //Solve the problem for each of the parameter set , for defined number of times.
            for(int j = 0; j<problemSolutionTimes; j++) {

                //Run continuous optimization algorithm to run for predefined maximum epochs (for each of the differential evolution mutation methods)
                ArrayList<DEIndividual> population1 = OptimizationBuilder.runContinuousOptimization_DErand1(problemB, populationSize, maxGenerations, F, crossoverRate)
                ArrayList<DEIndividual> population2 = OptimizationBuilder.runContinuousOptimization_DErand2(problemB, populationSize, maxGenerations, F, crossoverRate)
                ArrayList<DEIndividual> population3 = OptimizationBuilder.runContinuousOptimization_DEbest1(problemB, populationSize, maxGenerations, F, crossoverRate)
                ArrayList<DEIndividual> population4 = OptimizationBuilder.runContinuousOptimization_DEbest2(problemB, populationSize, maxGenerations, F, crossoverRate)
                ArrayList<DEIndividual> population5 = OptimizationBuilder.runContinuousOptimization_DEcurToBest1(problemB, populationSize, maxGenerations, F, crossover

                //Calculate the fitness levels for each of the mutation methods after maximum epochs reached.
                double fitness1 = DEFitness.getAverageDEPopulationFitness(population1);
                double fitness2 = DEFitness.getAverageDEPopulationFitness(population2);
                double fitness3 = DEFitness.getAverageDEPopulationFitness(population3);
                double fitness4 = DEFitness.getAverageDEPopulationFitness(population4);
                double fitness5 = DEFitness.getAverageDEPopulationFitness(population5);

                //Save the fitness scores to the different arraylists
                resultsAvgFitness1.add(fitness1);
                resultsAvgFitness2.add(fitness2);
                resultsAvgFitness3.add(fitness3);
                resultsAvgFitness4.add(fitness4);
                resultsAvgFitness5.add(fitness5);

            }

            //Calculate the metrics for the first mutation method : DE/RAND/1 and save them to the corresponding multi-metrics array
            double average = MathSupport.average(resultsAvgFitness1);
            double bestResult = MathSupport.getBestResult(resultsAvgFitness1);
            double worstResult = MathSupport.getWorstResult(resultsAvgFitness1);
            double median = MathSupport.median(resultsAvgFitness1);
            double stdDev = MathSupport.sd(resultsAvgFitness1);

            ArrayList<Double> metrics1 = new ArrayList<Double>();
            metrics1.add(average);
            metrics1.add(stdDev);
            metrics1.add(bestResult);
            metrics1.add(worstResult);
            metrics1.add(median);

            multiMetrics1.add(metrics1);
            multiResultsAvg1.add(resultsAvgFitness1);
```

```
            //Calculate the metrics for the first mutation method : DE/RAND/2 and save them to the corresponding multi-metrics array
            average = MathSupport.average(resultsAvgFitness2);
            bestResult = MathSupport.getBestResult(resultsAvgFitness2);
            worstResult = MathSupport.getWorstResult(resultsAvgFitness2);
            median = MathSupport.median(resultsAvgFitness2);
            stdDev = MathSupport.sd(resultsAvgFitness2);

            ArrayList<Double> metrics2 = new ArrayList<Double>();
            metrics2.add(average);
            metrics2.add(stdDev);
            metrics2.add(bestResult);
            metrics2.add(worstResult);
            metrics2.add(median);

            multiMetrics2.add(metrics2);
            multiResultsAvg2.add(resultsAvgFitness2);


            //Calculate the metrics for the first mutation method : DE/BEST/1 and save them to the corresponding multi-metrics array
            average = MathSupport.average(resultsAvgFitness3);
            bestResult = MathSupport.getBestResult(resultsAvgFitness3);
            worstResult = MathSupport.getWorstResult(resultsAvgFitness3);
            median = MathSupport.median(resultsAvgFitness3);
            stdDev = MathSupport.sd(resultsAvgFitness3);

            ArrayList<Double> metrics3 = new ArrayList<Double>();
            metrics3.add(average);
            metrics3.add(stdDev);
            metrics3.add(bestResult);
            metrics3.add(worstResult);
            metrics3.add(median);

            multiMetrics3.add(metrics3);
            multiResultsAvg3.add(resultsAvgFitness3);


            //Calculate the metrics for the first mutation method : DE/BEST/2 and save them to the corresponding multi-metrics array
            average = MathSupport.average(resultsAvgFitness4);
            bestResult = MathSupport.getBestResult(resultsAvgFitness4);
            worstResult = MathSupport.getWorstResult(resultsAvgFitness4);
            median = MathSupport.median(resultsAvgFitness4);
            stdDev = MathSupport.sd(resultsAvgFitness4);

            ArrayList<Double> metrics4 = new ArrayList<Double>();
            metrics4.add(average);
            metrics4.add(stdDev);
            metrics4.add(bestResult);
            metrics4.add(worstResult);
            metrics4.add(median);

            multiMetrics4.add(metrics4);
            multiResultsAvg4.add(resultsAvgFitness4);









            //Calculate the metrics for the first mutation method : DE/CURRENT_TO_BEST/1 and save them to the corresponding multi-metrics array
            average = MathSupport.average(resultsAvgFitness5);
            bestResult = MathSupport.getBestResult(resultsAvgFitness5);
            worstResult = MathSupport.getWorstResult(resultsAvgFitness5);
            median = MathSupport.median(resultsAvgFitness5);
            stdDev = MathSupport.sd(resultsAvgFitness5);

            ArrayList<Double> metrics5 = new ArrayList<Double>();
            metrics5.add(average);
            metrics5.add(stdDev);
            metrics5.add(bestResult);
            metrics5.add(worstResult);
            metrics5.add(median);

            multiMetrics5.add(metrics5);
            multiResultsAvg5.add(resultsAvgFitness5);


        }

        //Show metrics for DE/RAND/1
        System.out.println("\n MUTATION METHOD -> DE/RAND/1 -> METRICS \n ");
        PrintSupport.printMultiMetrics(problemB, multiResultsAvg1, multiMetrics1);

        //Show metrics for DE/RAND/2
        System.out.println("\n MUTATION METHOD -> DE/RAND/2 -> METRICS \n ");
        PrintSupport.printMultiMetrics(problemB, multiResultsAvg2, multiMetrics2);

        //Show metrics for DE/BEST/1
        System.out.println("\n MUTATION METHOD -> DE/BEST/1 -> METRICS \n ");
        PrintSupport.printMultiMetrics(problemB, multiResultsAvg3, multiMetrics3);

        //Show metrics for DE/BEST/2
        System.out.println("\n MUTATION METHOD -> DE/BEST/2 -> METRICS \n ");
        PrintSupport.printMultiMetrics(problemB, multiResultsAvg4, multiMetrics4);

        //Show metrics for DE/CURRENT_TO_BEST/1
        System.out.println("\n MUTATION METHOD -> DE/CURRENT_TO_BEST/1 -> METRICS \n ");
        PrintSupport.printMultiMetrics(problemB, multiResultsAvg5, multiMetrics5);

    }

}
```

(GENETİK ALGORİTMA TEST)

```java
public class GAGeneralTest {

    public static void main(String[] args) {


        /*
         *  THIS CLASS IS USED FOR TESTING THE USE OF A CONTINUOUS OPTIMIZATION ALGORITHM FOR OPTIMIZING
         *  THE PROBLEM THAT WAS MENTIONED IN "HOMEWORK PART 2 : CITY A"
         *
         *  USED METHOD : GENETIC ALGORITHM
         *
         *  SELECTION WORKS AS THE FOLLOWING :
         *
         *  -> AN ELITE GROUP IS SELECTED AMONG THE POPULATION EACH EPOCH, REGARDING THEIR FITNESS SCORES.
         *  -> A SPECIFIC PERCENT OF THIS ELITE GROUP TRANSFERS TO THE NEXT GENERATION DIRECTLY, AS ELDERS.
         *  -> BY USING THE ELITE GROUP'S GENES, SPECIFIC PERCENTAGE OF CHILDREN ARE CREATED, AS WELL AS BEING ABLE TO APPLY A CROSSING-OVER; PASSING TO OTHER GE
         *  -> A SPECIFIC PERCENT OF RANDOM INDIVIDUALS CAN BE SELECTED FROM THE POPULATION TO PASS TO OTHER GENERATION
         *  -> MUTATION CAN APPLY, CONTROLLED BY THE MUTATION RATE
         *
         *  IN THIS CLASS, YOU CAN EVALUATE THE PERFORMANCE OF THE GENETIC ALGORITHM
         *
         */

        //Genetic Algorithm Hyper-parameters
        int populationSize = 100;
        int maxGenerations = 2000;
        double elitismAmount = (populationSize*0.4);
        double elderAmount = (populationSize*0*1);
        double childrenAmount = (populationSize*0.89);
        double randomIndividualAmount = (populationSize*0.01);
        double crossoverRate = 0.9;
        double mutationRate = 0.1;


        //Multi-Solution Hyper-parameters
        double parameterRandomizationTimes = 4;
        double problemSolutionTimes = 25;


        //Define the city and the problem
        Problem problemA = GlobalGAVariables.problemA;




         //Define the multi-metrics array for different parameter sets and solution metrics for the algorithm
         ArrayList<ArrayList<Double>> multiMetrics = new ArrayList<ArrayList<Double>>();
         ArrayList<ArrayList<Double>> multiResultsAvg = new ArrayList<ArrayList<Double>>();

         //Run the genetic algorithm for a predefined number of times with different parameter sets
         for(int i = 0; i<parameterRandomizationTimes; i++) {

             ArrayList<Double> resultsAvgFitness = new ArrayList<Double>();

             //Solve the problem for a predefined number of times
             for(int j = 0; j<problemSolutionTimes; j++) {

                 ArrayList<Individual> population = OptimizationBuilder.runGA(problemA, populationSize, maxGenerations,
                         elitismAmount, elderAmount, randomIndividualAmount,
                         childrenAmount, crossoverRate, mutationRate);

                 double fitness = Fitness.getAveragePopulationFitness(population);
                 resultsAvgFitness.add(fitness);

             }

             //Calculate the metrics for the solutions and save them to the multimetrics array
             double average = MathSupport.average(resultsAvgFitness);
             double bestResult = MathSupport.getBestResult(resultsAvgFitness);
             double worstResult = MathSupport.getWorstResult(resultsAvgFitness);
             double median = MathSupport.median(resultsAvgFitness);
             double stdDev = MathSupport.sd(resultsAvgFitness);

             ArrayList<Double> metrics = new ArrayList<Double>();
             metrics.add(average);
             metrics.add(stdDev);
             metrics.add(bestResult);
             metrics.add(worstResult);
             metrics.add(median);

             multiMetrics.add(metrics);
             multiResultsAvg.add(resultsAvgFitness);
         }


         //Show metrics for the Genetic Algorithm solution
         System.out.println("\n GENETIC ALGORITHM SOLUTION METRICS : \n");
         PrintSupport.printMultiMetrics(problemA, multiResultsAvg, multiMetrics);
    }

}
```

(DE/RAND/1 TEST)

```java
public class COTest_DErand1 {

    public static void main(String[] args) {

        /*
         *  THIS CLASS IS USED FOR TESTING THE USE OF A CONTINUOUS OPTIMIZATION ALGORITHM FOR OPTIMIZING
         *  THE PROBLEM THAT WAS MENTIONED IN "HOMEWORK PART 3 : CITY B"
         *
         *  USED METHOD : DIFFERENTIAL EVOLUTION
         *
         *  IN THIS CLASS , THE MUTATION METHOD IS :
         *  -> DE/RAND/1
         *
         *  BY USING THIS PAGE, YOU CAN TEST THE PERFORMANCE OF THIS MUTATION METHOD SEPERATELY.
         *
         */

        //Continuous Optimization Hyper-parameters
        int populationSize = 100;
        int maxGenerations = 1000;
        double F = 0.1;
        double crossoverRate = 0.9;


        //Multi-Solution Hyper-parameters
        double parameterRandomizationTimes = 4;
        double problemSolutionTimes = 25;


        //Achieve the problem from the global variables class
        Problem problemB = GlobalDEVariables.problemB;
        problemB.showCustomerCoordinates();


        //Define the arrraylists that will hold the multi-metrics for different parameter sets and solutions
        ArrayList<ArrayList<Double>> multiMetrics = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiResultsAvg = new ArrayList<ArrayList<Double>>();







        //Run the algorithm for a predefined number of different parameter sets
        for(int i = 0; i<parameterRandomizationTimes; i++) {

            ArrayList<Double> resultsAvgFitness = new ArrayList<Double>();

            //Solve the problem for a predefined number of times
            for(int j = 0; j<problemSolutionTimes; j++) {

                ArrayList<DEIndividual> population = OptimizationBuilder.runContinuousOptimization_DErand1(problemB, populationSize, maxGenerations, F, crossoverRate);

                double fitness = DEFitness.getAverageDEPopulationFitness(population);
                resultsAvgFitness.add(fitness);

            }

            //Calculate the metrics and save them to the multi-metrics array
            double average = MathSupport.average(resultsAvgFitness);
            double bestResult = MathSupport.getBestResult(resultsAvgFitness);
            double worstResult = MathSupport.getWorstResult(resultsAvgFitness);
            double median = MathSupport.median(resultsAvgFitness);
            double stdDev = MathSupport.sd(resultsAvgFitness);

            ArrayList<Double> metrics = new ArrayList<Double>();
            metrics.add(average);
            metrics.add(stdDev);
            metrics.add(bestResult);
            metrics.add(worstResult);
            metrics.add(median);

            multiMetrics.add(metrics);
            multiResultsAvg.add(resultsAvgFitness);
        }

        //Show metrics for mutation method : DE/RAND/1
        System.out.println("\n MUTATION METHOD -> DE/RAND/1 -> METRICS \n ");
        PrintSupport.printMultiMetrics(problemB, multiResultsAvg, multiMetrics);

    }

}
```

(DE/RAND/2 TEST)

```java
public class COTest_DErand2 {

    public static void main(String[] args) {

        /*
         *  THIS CLASS IS USED FOR TESTING THE USE OF A CONTINUOUS OPTIMIZATION ALGORITHM FOR OPTIMIZING
         *  THE PROBLEM THAT WAS MENTIONED IN "HOMEWORK PART 3 : CITY B"
         *
         *  USED METHOD : DIFFERENTIAL EVOLUTION
         *
         *  IN THIS CLASS , THE MUTATION METHOD IS :
         *  -> DE/RAND/2
         *
         *  BY USING THIS PAGE, YOU CAN TEST THE PERFORMANCE OF THIS MUTATION METHOD SEPERATELY.
         *
         */

        //Continuous Optimization Hyper-parameters
        int populationSize = 100;
        int maxGenerations = 1000;
        double F = 0.1;
        double crossoverRate = 0.9;


        //Multi-Solution Hyper-parameters
        double parameterRandomizationTimes = 4;
        double problemSolutionTimes = 25;


        //Achieve the problem from the global variables class
        Problem problemB = GlobalDEVariables.problemB;
        problemB.showCustomerCoordinates();


        //Define the multi-metrics array for different parameter sets and solutions
        ArrayList<ArrayList<Double>> multiMetrics = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiResultsAvg = new ArrayList<ArrayList<Double>>();
```

```java
        //Run the continuous optimization algorithm for a predefined number of different parameter sets
        for(int i = 0; i<parameterRandomizationTimes; i++) {

            ArrayList<Double> resultsAvgFitness = new ArrayList<Double>();

            //Solve the problem for a predefinend number of times
            for(int j = 0; j<problemSolutionTimes; j++) {

                ArrayList<DEIndividual> population = OptimizationBuilder.runContinuousOptimization_DErand2(problemB, populationSize, maxGenerations, F, crossoverRate);

                double fitness = DEFitness.getAverageDEPopulationFitness(population);
                resultsAvgFitness.add(fitness);

            }

            //Calculate the metrics for solutions and save them to the corresponding multi-metrics array
            double average = MathSupport.average(resultsAvgFitness);
            double bestResult = MathSupport.getBestResult(resultsAvgFitness);
            double worstResult = MathSupport.getWorstResult(resultsAvgFitness);
            double median = MathSupport.median(resultsAvgFitness);
            double stdDev = MathSupport.sd(resultsAvgFitness);

            ArrayList<Double> metrics = new ArrayList<Double>();
            metrics.add(average);
            metrics.add(stdDev);
            metrics.add(bestResult);
            metrics.add(worstResult);
            metrics.add(median);

            multiMetrics.add(metrics);
            multiResultsAvg.add(resultsAvgFitness);
        }

        //Show metrics for mutation method : DE/RAND/2
        System.out.println("\n MUTATION METHOD -> DE/RAND/2 -> METRICS \n ");
        PrintSupport.printMultiMetrics(problemB, multiResultsAvg, multiMetrics);
    }

}
```

(DE/BEST/1 TEST)

```java
public class COTest_DEbest1 {

    public static void main(String[] args) {

        /*
         *  THIS CLASS IS USED FOR TESTING THE USE OF A CONTINUOUS OPTIMIZATION ALGORITHM FOR OPTIMIZING
         *  THE PROBLEM THAT WAS MENTIONED IN "HOMEWORK PART 3 : CITY B"
         *
         *  USED METHOD : DIFFERENTIAL EVOLUTION
         *
         *  IN THIS CLASS , THE MUTATION METHOD IS :
         *  -> DE/BEST/1
         *
         *  BY USING THIS PAGE, YOU CAN TEST THE PERFORMANCE OF THIS MUTATION METHOD SEPERATELY.
         *
         */


        //Continuous Optimization Hyper-parameters
        int populationSize = 100;
        int maxGenerations = 1000;
        double F = 0.1;
        double crossoverRate = 0.9;


        //Multi-Solution Hyper-parameters
        double parameterRandomizationTimes = 4;
        double problemSolutionTimes = 25;


        //achieve the problem from global variables class
        Problem problemB = GlobalDEVariables.problemB;
        problemB.showCustom
                tests.GlobalDEVariables
        Press 'F2' for focus


        //Define the multi-                    g the metrics for all of the calculations for different parameter sets
        ArrayList<ArrayList                    = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiResultsAvg = new ArrayList<ArrayList<Double>>();



       //Run the continuous optimization algorithm with different parameter sets for predefined number of times
       for(int i = 0; i<parameterRandomizationTimes; i++) {

           ArrayList<Double> resultsAvgFitness = new ArrayList<Double>();

           //Solve each parameter set for a predefined number of times
           for(int j = 0; j<problemSolutionTimes; j++) {

               //Run the differential evolution algorithm with mutation method : DE/BEST/1 : for predefined number of epochs
               ArrayList<DEIndividual> population = OptimizationBuilder.runContinuousOptimization_DEbest1(problemB, populationSize, maxGenerations, F, crossoverRate);

               double fitness = DEFitness.getAverageDEPopulationFitness(population);
               resultsAvgFitness.add(fitness);

           }

           //Calculate the metrics and save them to the multi-metrics array
           double average = MathSupport.average(resultsAvgFitness);
           double bestResult = MathSupport.getBestResult(resultsAvgFitness);
           double worstResult = MathSupport.getWorstResult(resultsAvgFitness);
           double median = MathSupport.median(resultsAvgFitness);
           double stdDev = MathSupport.sd(resultsAvgFitness);

           ArrayList<Double> metrics = new ArrayList<Double>();
           metrics.add(average);
           metrics.add(stdDev);
           metrics.add(bestResult);
           metrics.add(worstResult);
           metrics.add(median);

           multiMetrics.add(metrics);
           multiResultsAvg.add(resultsAvgFitness);
       }

       //Show metrics for mutation method : DE/BEST/1
       System.out.println("\n MUTATION METHOD -> DE/BEST/1 -> METRICS \n ");
       PrintSupport.printMultiMetrics(problemB, multiResultsAvg, multiMetrics);

    }

}
```

(DE/BEST/2 TEST)

```
public class COTest_DEbest2 {

    public static void main(String[] args) {

        /*
         *  THIS CLASS IS USED FOR TESTING THE USE OF A CONTINUOUS OPTIMIZATION ALGORITHM FOR OPTIMIZING
         *  THE PROBLEM THAT WAS MENTIONED IN "HOMEWORK PART 3 : CITY B"
         *
         *  USED METHOD : DIFFERENTIAL EVOLUTION
         *
         *  IN THIS CLASS , THE MUTATION METHOD IS :
         *  -> DE/BEST/2
         *
         *  BY USING THIS PAGE, YOU CAN TEST THE PERFORMANCE OF THIS MUTATION METHOD SEPERATELY.
         *
         */

        //Continuous Optimization Hyper-parameters
        int populationSize = 100;
        int maxGenerations = 1000;
        double F = 0.1;
        double crossoverRate = 0.9;


        //Multi-Solution Hyper-parameters
        double parameterRandomizationTimes = 4;
        double problemSolutionTimes = 25;


        //Achieve the problem from the global variables class
        Problem problemB = GlobalDEVariables.problemB;
        problemB.showCustomerCoordinates();


        //Define the array lists for holding the multi-metrics for different parameter sets and different solutions
        ArrayList<ArrayList<Double>> multiMetrics = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiResultsAvg = new ArrayList<ArrayList<Double>>();









       //Run the continuous optimization algorithm for predefined number of different parameter sets.
       for(int i = 0; i<parameterRandomizationTimes; i++) {

           ArrayList<Double> resultsAvgFitness = new ArrayList<Double>();

           //solve the problem for a predefined number of times
           for(int j = 0; j<problemSolutionTimes; j++) {

               ArrayList<DEIndividual> population = OptimizationBuilder.runContinuousOptimization_DEbest2(problemB, populationSize, maxGenerations, F, crossoverRate);

               double fitness = DEFitness.getAverageDEPopulationFitness(population);
               resultsAvgFitness.add(fitness);

           }

           //Calculate the metrics for the solutions and save them to the multi-metrics arraylist
           double average = MathSupport.average(resultsAvgFitness);
           double bestResult = MathSupport.getBestResult(resultsAvgFitness);
           double worstResult = MathSupport.getWorstResult(resultsAvgFitness);
           double median = MathSupport.median(resultsAvgFitness);
           double stdDev = MathSupport.sd(resultsAvgFitness);

           ArrayList<Double> metrics = new ArrayList<Double>();
           metrics.add(average);
           metrics.add(stdDev);
           metrics.add(bestResult);
           metrics.add(worstResult);
           metrics.add(median);

           multiMetrics.add(metrics);
           multiResultsAvg.add(resultsAvgFitness);
       }

       //Show metrics for mutation method : DE/BEST/2
       System.out.println("\n MUTATION METHOD -> DE/BEST/2 -> METRICS \n ");
       PrintSupport.printMultiMetrics(problemB, multiResultsAvg, multiMetrics);

   }

}
```

(DE/CURRENT_TO_BEST/1 TEST)

```java
public class COTest_DEcurToBest1 {

    public static void main(String[] args) {

        /*
         *  THIS CLASS IS USED FOR TESTING THE USE OF A CONTINUOUS OPTIMIZATION ALGORITHM FOR OPTIMIZING
         *  THE PROBLEM THAT WAS MENTIONED IN "HOMEWORK PART 3 : CITY B"
         *
         *  USED METHOD : DIFFERENTIAL EVOLUTION
         *
         *  IN THIS CLASS , THE MUTATION METHOD IS :
         *  -> DE/CURRENT_TO_BEST/1
         *
         *  BY USING THIS PAGE, YOU CAN TEST THE PERFORMANCE OF THIS MUTATION METHOD SEPERATELY.
         *
         */


        //Continuous Optimization Hyper-parameters
        int populationSize = 100;
        int maxGenerations = 1000;
        double F = 0.1;
        double crossoverRate = 0.9;


        //Multi-Solution Hyper-parameters
        double parameterRandomizationTimes = 4;
        double problemSolutionTimes = 25;


        //Achieve the problem from the global variables class
        Problem problemB = GlobalDEVariables.problemB;
        problemB.showCustomerCoordinates();


        //Define the array lists to hold the multi-metrics for different solutions and different parameter sets
        ArrayList<ArrayList<Double>> multiMetrics = new ArrayList<ArrayList<Double>>();
        ArrayList<ArrayList<Double>> multiResultsAvg = new ArrayList<ArrayList<Double>>();




       //Run the continuous optimization for a predefined number of different parameter sets
       for(int i = 0; i<parameterRandomizationTimes; i++) {

           ArrayList<Double> resultsAvgFitness = new ArrayList<Double>();

           //Solve the problem for a predefined number of times
           for(int j = 0; j<problemSolutionTimes; j++) {

               ArrayList<DEIndividual> population = OptimizationBuilder.runContinuousOptimization_DEcurToBest1(problemB, populationSize, maxGenerations, F, crossover

               double fitness = DEFitness.getAverageDEPopulationFitness(population);
               resultsAvgFitness.add(fitness);

           }

           //Calculate the metrics for the calculations and add them to the multi-metrics array
           double average = MathSupport.average(resultsAvgFitness);
           double bestResult = MathSupport.getBestResult(resultsAvgFitness);
           double worstResult = MathSupport.getWorstResult(resultsAvgFitness);
           double median = MathSupport.median(resultsAvgFitness);
           double stdDev = MathSupport.sd(resultsAvgFitness);

           ArrayList<Double> metrics = new ArrayList<Double>();
           metrics.add(average);
           metrics.add(stdDev);
           metrics.add(bestResult);
           metrics.add(worstResult);
           metrics.add(median);

           multiMetrics.add(metrics);
           multiResultsAvg.add(resultsAvgFitness);
       }

       //Show metrics for mutation method : DE/CURRENT_TO_BEST/1
       System.out.println("\n MUTATION METHOD -> DE/CURRENT_TO_BEST/1 -> METRICS \n ");
       PrintSupport.printMultiMetrics(problemB, multiResultsAvg, multiMetrics);

    }

}
```

(GLOBAL DEĞİŞKENLER)

```java
package tests;

import model.City;

public class GlobalGAVariables {

    /*
     *  THESE VARIABLES DEFINE THE "CITY A" THAT WAS MENTIONED IN THE HOMEWORK (PART : 2)
     *
     * */

    //Problem parameters
    private static double width = 400;
    private static double height = 400;
    private static int customerAmount = 250;
    private static int baseLocationAmount = 100;
    private static double coverRadius = 30;
    private static int baseAmount = 50;

    //Defining the city and the problem for CITY A
    public static City cityA = new City(width, height, customerAmount, baseLocationAmount, baseAmount);
    public static Problem problemA = new Problem(cityA, coverRadius);
}
```

```java
package tests;

import model.City;

public class GlobalDEVariables {

    /*
     *  THESE VARIABLES DEFINE THE "CITY B" THAT WAS MENTIONED IN THE HOMEWORK (PART : 3)
     *
     * */


    //Problem parameters
    private static double width = 500;
    private static double height = 500;
    private static int customerAmount = 300;
    private static int baseLocationAmount = 0;
    private static double coverRadius = 35;
    private static int baseAmount = 50;

    //Defining the City and Problem Objects for CITY B
    public static City cityB = new City(width, height, customerAmount, baseLocationAmount, baseAmount);
    public static Problem problemB = new Problem(cityB, coverRadius);

}
```

(PROBLEM TANIMI TEST)

```java

package tests;

import model.Problem;

public class ProblemDefinitionsTest {

    /*
     *
     * THIS CLASS IS USED FOR TESTING THE PRINTING OF THE PROBLEM DEFINITIONS WHICH WAS MENTIONED ON HOMEWORK -> PART 1
     *
     */

    public static void main(String[] args) {

        //Achieving PROBLEM A -> solves the problem in CITY A
        Problem problemA = GlobalGAVariables.problemA;
        problemA.toString();

        //Achieving PROBLEM B -> solved the problem in CITY B
        Problem problemB = GlobalDEVariables.problemB;
        problemB.toString();

    }
}

```

4.Çıktı / Output Görüntüleri

(PROBLEM TANIMI – BÖLÜM 1)

```
 PROBLEM – CUSTOMERS AND BASE LOCATIONS :

 CUSTOMER LOCATIONS (X / Y) :
Customer 0 (186,29 / 118,64) ,
Customer 1 (60,22 / 370,40) ,
Customer 2 (331,42 / 227,87) ,
Customer 3 (285,61 / 336,38) ,
Customer 4 (152,12 / 254,21) ,
Customer 5 (77,60 / 335,36) ,
Customer 6 (173,73 / 151,96) ,
Customer 7 (287,21 / 96,60) ,
Customer 8 (217,07 / 109,30) ,
Customer 9 (316,06 / 214,52) ,
Customer 10 (278,94 / 1,38) ,
Customer 11 (194,91 / 92,77) ,
Customer 12 (304,78 / 160,73) ,
Customer 13 (72,96 / 174,96) ,
Customer 14 (203,53 / 317,35) ,
Customer 15 (376,47 / 237,22) ,
Customer 16 (362,82 / 322,28) ,
Customer 17 (35,17 / 360,49) ,
Customer 18 (393,32 / 92,13) ,
Customer 19 (276,35 / 105,51) ,
Customer 20 (264,41 / 298,52) ,
Customer 21 (351,47 / 380,75) ,
Customer 22 (188,89 / 75,11) ,
Customer 23 (386,69 / 324,20) ,
Customer 24 (372,69 / 226,34) ,
Customer 25 (138,40 / 211,89) ,
Customer 26 (266,32 / 95,63) ,
Customer 27 (386,76 / 161,51) ,
Customer 28 (37,94 / 187,31) ,
Customer 29 (14,61 / 16,84) ,
Customer 30 (233,52 / 35,37) ,
Customer 31 (96,15 / 278,98) ,
Customer 32 (198,08 / 73,27) ,
Customer 33 (159,26 / 92,70) ,
Customer 34 (202,37 / 183,06) ,
Customer 35 (387,15 / 101,65) ,
Customer 36 (352,68 / 188,33) ,
Customer 37 (89,66 / 253,29) ,
Customer 38 (57,26 / 4,32) ,
Customer 39 (101,44 / 164,94) ,
Customer 40 (280,89 / 269,84) ,
Customer 41 (53,44 / 54,62) ,
Customer 42 (239,96 / 59,55) ,
Customer 43 (316,64 / 295,59) ,
Customer 44 (173,50 / 311,34) ,
Customer 45 (30,68 / 170,97) ,
Customer 46 (371,46 / 302,56) ,
Customer 47 (207,80 / 184,94) ,
Customer 48 (149,56 / 270,06) ,
Customer 49 (41,17 / 129,64) ,
Customer 50 (130,92 / 92,78) ,
Customer 51 (49,30 / 211,95) ,
Customer 52 (227,75 / 242,09) ,
Customer 53 (352,23 / 111,85) ,
Customer 54 (112,10 / 153,91) ,
Customer 55 (202,39 / 74,13) ,
Customer 56 (322,94 / 387,73) ,
Customer 57 (282,35 / 227,98) ,
Customer 58 (77,52 / 11,82) ,
Customer 59 (233,73 / 83,84) ,
Customer 60 (338,09 / 390,03) ,
Customer 61 (395,94 / 374,58) ,
Customer 62 (130,80 / 144,40) ,
```

Müşteri koordinatları.

```
Customer 211 (59,84 / 45,00) ,
Customer 212 (94,48 / 206,75) ,
Customer 213 (48,55 / 0,45) ,
Customer 214 (317,22 / 322,16) ,
Customer 215 (73,62 / 364,64) ,
Customer 216 (225,74 / 123,49) ,
Customer 217 (209,80 / 166,32) ,
Customer 218 (148,02 / 68,07) ,
Customer 219 (13,71 / 334,04) ,
Customer 220 (207,42 / 57,27) ,
Customer 221 (370,93 / 305,38) ,
Customer 222 (94,63 / 203,15) ,
Customer 223 (186,79 / 51,14) ,
Customer 224 (78,31 / 358,26) ,
Customer 225 (272,29 / 210,69) ,
Customer 226 (386,44 / 374,52) ,
Customer 227 (178,41 / 39,37) ,
Customer 228 (223,58 / 42,84) ,
Customer 229 (296,47 / 336,57) ,
Customer 230 (384,87 / 114,69) ,
Customer 231 (69,37 / 53,34) ,
Customer 232 (286,04 / 117,42) ,
Customer 233 (47,68 / 70,50) ,
Customer 234 (49,08 / 49,67) ,
Customer 235 (376,83 / 93,93) ,
Customer 236 (266,59 / 278,99) ,
Customer 237 (315,90 / 330,75) ,
Customer 238 (213,69 / 262,83) ,
Customer 239 (165,43 / 190,45) ,
Customer 240 (384,15 / 75,51) ,
Customer 241 (121,33 / 97,58) ,
Customer 242 (98,59 / 347,35) ,
Customer 243 (73,76 / 7,69) ,
Customer 244 (365,45 / 108,68) ,
Customer 245 (234,27 / 185,64) ,
Customer 246 (171,22 / 180,14) ,
Customer 247 (301,60 / 235,77) ,
Customer 248 (134,87 / 162,10) ,
Customer 249 (265,87 / 65,66) ,
```

Müşteri koordinatları…

```
 BASE LOCATION COORDINATES (X / Y) :
Base Location 0 (19,63 / 344,43) ,
Base Location 1 (19,24 / 365,10) ,
Base Location 2 (3,90 / 83,12) ,
Base Location 3 (343,35 / 286,77) ,
Base Location 4 (47,53 / 184,46) ,
Base Location 5 (267,95 / 91,42) ,
Base Location 6 (330,75 / 382,52) ,
Base Location 7 (204,72 / 260,69) ,
Base Location 8 (375,05 / 147,64) ,
Base Location 9 (376,37 / 87,98) ,
Base Location 10 (218,85 / 277,66) ,
Base Location 11 (78,00 / 323,02) ,
Base Location 12 (57,29 / 77,75) ,
Base Location 13 (93,80 / 93,64) ,
Base Location 14 (169,01 / 190,94) ,
Base Location 15 (93,02 / 369,37) ,
Base Location 16 (320,41 / 136,93) ,
Base Location 17 (377,96 / 258,35) ,
Base Location 18 (67,94 / 375,48) ,
Base Location 19 (367,96 / 18,66) ,
Base Location 20 (71,69 / 30,75) ,
Base Location 21 (277,26 / 216,97) ,
Base Location 22 (88,29 / 96,68) ,
Base Location 23 (381,04 / 381,97) ,
Base Location 24 (129,14 / 388,93) ,
Base Location 25 (62,49 / 376,91) ,
Base Location 26 (149,68 / 317,11) ,
Base Location 27 (95,51 / 355,22) ,
Base Location 28 (178,30 / 321,94) ,
Base Location 29 (340,92 / 164,77) ,
Base Location 30 (198,32 / 154,88) ,
Base Location 31 (21,02 / 31,22) ,
Base Location 32 (164,17 / 73,76) ,
Base Location 33 (185,56 / 77,52) ,
Base Location 34 (63,12 / 285,93) ,
Base Location 35 (293,99 / 326,24) ,
Base Location 36 (326,02 / 191,26) ,
Base Location 37 (212,39 / 83,35) ,
Base Location 38 (146,20 / 160,53) ,
Base Location 39 (159,88 / 16,46) ,
Base Location 40 (182,87 / 330,36) ,
Base Location 41 (238,33 / 217,56) ,
Base Location 42 (305,37 / 4,19) ,
Base Location 43 (140,11 / 279,66) ,
Base Location 44 (77,60 / 119,33) ,
Base Location 45 (52,90 / 118,82) ,
Base Location 46 (386,64 / 273,48) ,
Base Location 47 (66,57 / 137,37) ,
Base Location 48 (187,28 / 199,66) ,
Base Location 49 (336,20 / 351,13) ,
Base Location 50 (364,43 / 136,90) ,
Base Location 51 (266,40 / 292,93) ,
Base Location 52 (7,66 / 64,54) ,
Base Location 53 (56,20 / 56,68) ,
Base Location 54 (147,36 / 251,22) ,
Base Location 55 (79,86 / 160,57) ,
Base Location 56 (219,71 / 215,58) ,
Base Location 57 (377,95 / 385,10) ,
Base Location 58 (7,36 / 3,53) ,
Base Location 59 (394,20 / 181,03) ,
Base Location 60 (145,81 / 257,89) ,
Base Location 61 (17,75 / 44,23) ,
Base Location 62 (289,98 / 289,43) ,
Base Location 63 (176,30 / 18,94) ,
```

Baz istasyonu yerleşim noktası koordinatları…

```
Base Location 55 (79,86 / 160,57) ,
Base Location 56 (219,71 / 215,58) ,
Base Location 57 (377,95 / 385,10) ,
Base Location 58 (7,36 / 3,53) ,
Base Location 59 (394,20 / 181,03) ,
Base Location 60 (145,81 / 257,89) ,
Base Location 61 (17,75 / 44,23) ,
Base Location 62 (289,98 / 289,43) ,
Base Location 63 (176,30 / 18,94) ,
Base Location 64 (103,61 / 253,82) ,
Base Location 65 (274,97 / 217,90) ,
Base Location 66 (363,72 / 197,48) ,
Base Location 67 (286,77 / 137,22) ,
Base Location 68 (65,26 / 355,78) ,
Base Location 69 (74,25 / 195,84) ,
Base Location 70 (66,08 / 70,42) ,
Base Location 71 (358,60 / 343,89) ,
Base Location 72 (129,79 / 238,42) ,
Base Location 73 (251,07 / 361,20) ,
Base Location 74 (204,33 / 313,37) ,
Base Location 75 (222,81 / 206,64) ,
Base Location 76 (16,63 / 350,41) ,
Base Location 77 (13,41 / 80,95) ,
Base Location 78 (368,63 / 219,32) ,
Base Location 79 (264,29 / 330,29) ,
Base Location 80 (87,85 / 306,23) ,
Base Location 81 (212,33 / 144,35) ,
Base Location 82 (15,40 / 382,23) ,
Base Location 83 (390,28 / 392,73) ,
Base Location 84 (107,13 / 238,19) ,
Base Location 85 (154,19 / 295,00) ,
Base Location 86 (78,77 / 368,12) ,
Base Location 87 (194,66 / 334,33) ,
Base Location 88 (43,10 / 229,11) ,
Base Location 89 (387,51 / 101,63) ,
Base Location 90 (67,87 / 214,34) ,
Base Location 91 (200,51 / 396,46) ,
Base Location 92 (125,05 / 245,84) ,
Base Location 93 (397,14 / 313,03) ,
Base Location 94 (107,82 / 27,48) ,
Base Location 95 (164,07 / 204,12) ,
Base Location 96 (258,62 / 294,00) ,
Base Location 97 (238,57 / 342,37) ,
Base Location 98 (282,16 / 50,06) ,
Base Location 99 (122,65 / 388,32) ,
```

Baz istasyonu yerleşim noktası koordinatları…

```
 CUSTOMER LOCATIONS (X / Y) :
Customer 0 (336,76 / 53,24) ,
Customer 1 (269,26 / 395,79) ,
Customer 2 (377,81 / 464,88) ,
Customer 3 (36,68 / 4,81) ,
Customer 4 (153,17 / 441,28) ,
Customer 5 (55,77 / 274,65) ,
Customer 6 (90,83 / 456,62) ,
Customer 7 (409,15 / 344,80) ,
Customer 8 (143,01 / 343,82) ,
Customer 9 (310,34 / 173,91) ,
Customer 10 (325,94 / 427,82) ,
Customer 11 (133,43 / 263,78) ,
Customer 12 (199,89 / 447,77) ,
Customer 13 (388,98 / 470,60) ,
Customer 14 (66,10 / 477,99) ,
Customer 15 (407,31 / 123,14) ,
Customer 16 (317,89 / 175,21) ,
Customer 17 (96,45 / 43,31) ,
Customer 18 (59,44 / 437,92) ,
Customer 19 (13,50 / 330,52) ,
Customer 20 (297,70 / 462,67) ,
Customer 21 (125,45 / 428,09) ,
Customer 22 (220,40 / 221,49) ,
Customer 23 (38,14 / 37,21) ,
Customer 24 (296,26 / 64,98) ,
Customer 25 (140,80 / 382,06) ,
Customer 26 (172,18 / 476,43) ,
Customer 27 (47,48 / 158,29) ,
Customer 28 (96,89 / 463,51) ,
Customer 29 (416,41 / 216,12) ,
Customer 30 (370,02 / 176,47) ,
Customer 31 (193,69 / 139,50) ,
Customer 32 (101,86 / 84,12) ,
Customer 33 (58,69 / 383,01) ,
Customer 34 (369,77 / 422,51) ,
Customer 35 (345,19 / 19,38) ,
Customer 36 (174,39 / 154,75) ,
Customer 37 (474,18 / 432,17) ,
Customer 38 (431,30 / 345,41) ,
Customer 39 (462,53 / 460,80) ,
Customer 40 (81,92 / 157,72) ,
Customer 41 (364,90 / 67,59) ,
Customer 42 (270,99 / 127,88) ,
Customer 43 (65,27 / 50,61) ,
Customer 44 (417,80 / 456,96) ,
Customer 45 (159,55 / 170,09) ,
Customer 46 (402,93 / 288,23) ,
Customer 47 (260,65 / 291,05) ,
Customer 48 (3,12 / 305,07) ,
Customer 49 (413,62 / 270,30) ,
Customer 50 (133,22 / 493,32) ,
Customer 51 (429,85 / 332,14) ,
Customer 52 (395,80 / 60,41) ,
Customer 53 (336,84 / 83,70) ,
Customer 54 (396,86 / 293,56) ,
Customer 55 (292,36 / 118,45) ,
Customer 56 (258,88 / 69,26) ,
Customer 57 (230,99 / 343,98) ,
Customer 58 (251,59 / 297,22) ,
Customer 59 (194,30 / 249,40) ,
Customer 60 (464,82 / 293,12) ,
Customer 61 (246,41 / 296,15) ,
```

Müşteri koordinatları…

```
Customer 244 (42,43 / 100,32) ,
Customer 245 (103,54 / 316,61) ,
Customer 246 (304,96 / 323,87) ,
Customer 247 (24,47 / 110,39) ,
Customer 248 (142,62 / 380,96) ,
Customer 249 (280,46 / 499,90) ,
Customer 250 (197,52 / 55,51) ,
Customer 251 (43,87 / 481,18) ,
Customer 252 (403,64 / 372,61) ,
Customer 253 (223,08 / 47,26) ,
Customer 254 (179,22 / 228,59) ,
Customer 255 (404,34 / 237,03) ,
Customer 256 (150,06 / 102,11) ,
Customer 257 (305,59 / 198,75) ,
Customer 258 (107,17 / 365,75) ,
Customer 259 (8,45 / 0,57) ,
Customer 260 (469,13 / 491,46) ,
Customer 261 (246,65 / 452,48) ,
Customer 262 (101,87 / 357,65) ,
Customer 263 (307,18 / 126,31) ,
Customer 264 (446,81 / 421,22) ,
Customer 265 (143,59 / 73,42) ,
Customer 266 (22,18 / 213,14) ,
Customer 267 (110,39 / 443,36) ,
Customer 268 (491,47 / 453,15) ,
Customer 269 (301,73 / 300,98) ,
Customer 270 (138,29 / 55,66) ,
Customer 271 (318,32 / 424,42) ,
Customer 272 (247,26 / 417,15) ,
Customer 273 (392,11 / 241,68) ,
Customer 274 (67,15 / 133,21) ,
Customer 275 (170,12 / 329,93) ,
Customer 276 (224,03 / 192,54) ,
Customer 277 (95,88 / 58,16) ,
Customer 278 (159,96 / 234,73) ,
Customer 279 (229,84 / 1,17) ,
Customer 280 (395,85 / 138,79) ,
Customer 281 (18,43 / 191,10) ,
Customer 282 (155,83 / 30,23) ,
Customer 283 (455,78 / 101,10) ,
Customer 284 (144,90 / 255,81) ,
Customer 285 (281,54 / 475,03) ,
Customer 286 (41,53 / 477,43) ,
Customer 287 (323,78 / 226,93) ,
Customer 288 (52,94 / 278,78) ,
Customer 289 (326,79 / 134,68) ,
Customer 290 (181,33 / 62,37) ,
Customer 291 (100,23 / 442,97) ,
Customer 292 (167,09 / 37,70) ,
Customer 293 (88,32 / 272,97) ,
Customer 294 (406,70 / 266,65) ,
Customer 295 (485,45 / 407,88) ,
Customer 296 (491,95 / 36,69) ,
Customer 297 (318,26 / 87,12) ,
Customer 298 (380,17 / 278,79) ,
Customer 299 (473,15 / 394,97) ,
```

Müşteri koordinatları…

```
This city does not have mandatory base locations! The problem is continuous!
```

(GENETİK ALGORİTMA HESAP VE SONUÇLARI – BÖLÜM 2)

```
_____________________
GENERATION :305
AVERAGE FITNESS OF POPULATION : 179.82222222222222 / 250 ( %71.92888888888889 )

_____________________
GENERATION :306
AVERAGE FITNESS OF POPULATION : 179.8111111111111 / 250 ( %71.92444444444445 )

_____________________
GENERATION :307
AVERAGE FITNESS OF POPULATION : 179.6 / 250 ( %71.83999999999999 )

_____________________
GENERATION :308
AVERAGE FITNESS OF POPULATION : 179.62222222222223 / 250 ( %71.8488888888889 )

_____________________
GENERATION :309
AVERAGE FITNESS OF POPULATION : 179.61111111111111 / 250 ( %71.84444444444445 )

_____________________
GENERATION :310
AVERAGE FITNESS OF POPULATION : 179.66666666666666 / 250 ( %71.86666666666667 )

_____________________
GENERATION :311
AVERAGE FITNESS OF POPULATION : 179.74
int populationSize - tests.GAGeneralTest.main(String[])

_____________________
GENERATION :312
AVERAGE FITNESS OF POPULATION : 179.66666666666666 / 250 ( %71.86666666666667 )

_____________________
GENERATION :313
AVERAGE FITNESS OF POPULATION : 179.5888888888889 / 250 ( %71.83555555555556 )

_____________________
GENERATION :314
AVERAGE FITNESS OF POPULATION : 179.75555555555556 / 250 ( %71.90222222222222 )

_____________________
GENERATION :315
AVERAGE FITNESS OF POPULATION : 179.7111111111111 / 250 ( %71.88444444444444 )

_____________________
GENERATION :316
AVERAGE FITNESS OF POPULATION : 179.82222222222222 / 250 ( %71.92888888888889 )

_____________________
GENERATION :317
```

Genetik algoritma çalışma sürecinden bir görüntü…

AVERAGE FITNESS OF POPULATION : 181.65555555555557 / 250 (%72.66222222222223)

GENERATION :1665

AVERAGE FITNESS OF POPULATION : 181.65555555555557 / 250 (%72.66222222222223)

GENERATION :1666

AVERAGE FITNESS OF POPULATION : 181.6888888888889 / 250 (%72.67555555555556)

GENERATION :1667

AVERAGE FITNESS OF POPULATION : 181.6888888888889 / 250 (%72.67555555555556)

GENERATION :1668

AVERAGE FITNESS OF POPULATION : 181.8 / 250 (%72.72000000000001)

GENERATION :1669

AVERAGE FITNESS OF POPULATION : 181.86

int populationSize - tests.GAGeneralTest.main(String[])

GENERATION :1670

AVERAGE FITNESS OF POPULATION : 181.56666666666666 / 250 (%72.62666666666667)

GENERATION :1671

AVERAGE FITNESS OF POPULATION : 181.7111111111111 / 250 (%72.68444444444444)

GENERATION :1672

AVERAGE FITNESS OF POPULATION : 181.64444444444445 / 250 (%72.65777777777778)

GENERATION :1673

AVERAGE FITNESS OF POPULATION : 181.72222222222223 / 250 (%72.6888888888889)

GENERATION :1674

Genetik algoritma çalışma sürecinden bir görüntü…

```
GENERATION :615
AVERAGE FITNESS OF POPULATION : 180.85555555555555 / 250 ( %72.34222222222222 )

_____________________
GENERATION :616
AVERAGE FITNESS OF POPULATION : 180.92222222222222 / 250 ( %72.36888888888888 )

_____________________
GENERATION :617
AVERAGE FITNESS OF POPULATION : 180.77777777777777 / 250 ( %72.3111111111111 )

_____________________
GENERATION :618
AVERAGE FITNESS OF POPULATION : 180.8 / 250 ( %72.32000000000001 )

_____________________
GENERATION :619
AVERAGE FITNESS OF POPULATION : 180.7 / 250 ( %72.28 )

_____________________
GENERATION :620
AVERAGE FITNESS OF POPULATION : 180.84

_____________________
GENERATION :621
AVERAGE FITNESS OF POPULATION : 180.64444444444445 / 250 ( %72.25777777777778 )

_____________________
GENERATION :622
AVERAGE FITNESS OF POPULATION : 180.77777777777777 / 250 ( %72.3111111111111 )

_____________________
GENERATION :623
AVERAGE FITNESS OF POPULATION : 180.6888888888889 / 250 ( %72.27555555555556 )

_____________________
GENERATION :624
AVERAGE FITNESS OF POPULATION : 180.83333333333334 / 250 ( %72.33333333333334 )

_____________________
GENERATION :625
AVERAGE FITNESS OF POPULATION : 180.73333333333332 / 250 ( %72.29333333333334 )
```

int populationSize - tests.GAGeneralTest.main(String[])

Genetik algoritma çalışma sürecinden bir görüntü…

```
 GENETIC ALGORITHM SOLUTION METRICS :

RESULTS FOR PARAMETER SET - 0 :
RUN 1 : 206.48888888888888 / 250 ( %82.59555555555555 )
RUN 2 : 205.7 / 250 ( %82.28 )
RUN 3 : 207.6 / 250 ( %83.04 )
RUN 4 : 207.4 / 250 ( %82.96 )
RUN 5 : 206.88888888888889 / 250 ( %82.75555555555555 )
RUN 6 : 206.63333333333333 / 250 ( %82.65333333333334 )
RUN 7 : 205.72222222222223 / 250 ( %82.28888888888889 )
RUN 8 : 203.73333333333332 / 250 ( %81.49333333333333 )
RUN 9 : 205.66666666666666 / 250 ( %82.26666666666667 )
RUN 10 : 207.7 / 250 ( %83.08 )
RUN 11 : 204.48888888888888 / 250 ( %81.79555555555555 )
RUN 12 : 206.77777777777777 / 250 ( %82.71111111111111 )
RUN 13 : 207.57777777777778 / 250 ( %83.03111111111112 )
RUN 14 : 206.57777777777778 / 250 ( %82.63111111111111 )
RUN 15 : 207.48888888888888 / 250 ( %82.99555555555555 )
RUN 16 : 206.57777777777778 / 250 ( %82.63111111111111 )
RUN 17 : 206.6 / 250 ( %82.64 )
RUN 18 : 204.7111111111111 / 250 ( %81.88444444444445 )
RUN 19 : 205.83333333333334 / 250 ( %82.33333333333334 )
RUN 20 : 206.77777777777777 / 250 ( %82.71111111111111 )
RUN 21 : 204.83333333333334 / 250 ( %81.93333333333334 )
RUN 22 : 206.56666666666666 / 250 ( %82.62666666666667 )
RUN 23 : 204.84444444444443 / 250 ( %81.93777777777778 )
RUN 24 : 206.6 / 250 ( %82.64 )
RUN 25 : 207.6888888888889 / 250 ( %83.07555555555555 )
_____________________
AVERAGE : 206.29911111111107 / 250 ( %82.51964444444442 )
STANDARD DEVIATION : 1.080104796013041 / 250 ( %0.4320419184052164 )
BEST RESULT : 207.7 / 250 ( %83.08 )
WORST RESULT : 203.73333333333332 / 250 ( %81.49333333333333 )
MEDIAN : 207.17777777777778 / 250 ( %82.8711111111111 )
**********************
```

1. Problem tanımı için sonuçlar

```
RESULTS FOR PARAMETER SET - 1 :
RUN 1 : 206.61111111111111 / 250 ( %82.64444444444445 )
RUN 2 : 206.64444444444445 / 250 ( %82.65777777777778 )
RUN 3 : 206.56666666666666 / 250 ( %82.62666666666667 )
RUN 4 : 205.61111111111111 / 250 ( %82.24444444444444 )
RUN 5 : 206.88888888888889 / 250 ( %82.75555555555555 )
RUN 6 : 205.57777777777778 / 250 ( %82.2311111111111 )
RUN 7 : 204.56666666666666 / 250 ( %81.82666666666667 )
RUN 8 : 206.72222222222223 / 250 ( %82.68888888888888 )
RUN 9 : 203.55555555555554 / 250 ( %81.42222222222222 )
RUN 10 : 207.66666666666666 / 250 ( %83.06666666666666 )
RUN 11 : 204.73333333333332 / 250 ( %81.89333333333333 )
RUN 12 : 206.75555555555556 / 250 ( %82.70222222222222 )
RUN 13 : 205.46666666666667 / 250 ( %82.18666666666667 )
RUN 14 : 206.66666666666666 / 250 ( %82.66666666666667 )
RUN 15 : 205.8 / 250 ( %82.32000000000001 )
RUN 16 : 207.76666666666668 / 250 ( %83.10666666666667 )
RUN 17 : 205.75555555555556 / 250 ( %82.30222222222223 )
RUN 18 : 206.67777777777778 / 250 ( %82.6711111111111 )
RUN 19 : 204.36666666666667 / 250 ( %81.74666666666667 )
RUN 20 : 206.48888888888888 / 250 ( %82.59555555555555 )
RUN 21 : 206.6 / 250 ( %82.64 )
RUN 22 : 206.66666666666666 / 250 ( %82.66666666666667 )
RUN 23 : 207.7111111111111 / 250 ( %83.08444444444444 )
RUN 24 : 206.5888888888889 / 250 ( %82.63555555555556 )
RUN 25 : 205.54444444444445 / 250 ( %82.21777777777778 )
_____
AVERAGE : 206.16 / 250 ( %82.464 )
STANDARD DEVIATION : 1.033373953801364 / 250 ( %0.41334958152054563 )
BEST RESULT : 207.76666666666668 / 250 ( %83.10666666666667 )
WORST RESULT : 203.55555555555554 / 250 ( %81.42222222222222 )
MEDIAN : 206.11111111111111 / 250 ( %82.44444444444444 )
**********************
```

2. Problem tanımı için sonuçlar

```
RESULTS FOR PARAMETER SET - 2 :
RUN 1 : 205.54444444444445 / 250 ( %82.21777777777778 )
RUN 2 : 205.9 / 250 ( %82.36 )
RUN 3 : 206.4777777777778 / 250 ( %82.59111111111112 )
RUN 4 : 205.8111111111111 / 250 ( %82.32444444444444 )
RUN 5 : 206.74444444444444 / 250 ( %82.69777777777779 )
RUN 6 : 207.6 / 250 ( %83.04 )
RUN 7 : 206.8 / 250 ( %82.72 )
RUN 8 : 205.55555555555554 / 250 ( %82.22222222222221 )
RUN 9 : 207.7111111111111 / 250 ( %83.08444444444444 )
RUN 10 : 207.85555555555555 / 250 ( %83.14222222222222 )
RUN 11 : 204.74444444444444 / 250 ( %81.89777777777778 )
RUN 12 : 205.5 / 250 ( %82.19999999999999 )
RUN 13 : 206.62222222222223 / 250 ( %82.64888888888889 )
RUN 14 : 205.5222222222222 / 250 ( %82.2088888888889 )
RUN 15 : 205.64444444444445 / 250 ( %82.25777777777778 )
RUN 16 : 206.73333333333332 / 250 ( %82.69333333333333 )
RUN 17 : 207.7 / 250 ( %83.08 )
RUN 18 : 207.7 / 250 ( %83.08 )
RUN 19 : 205.75555555555556 / 250 ( %82.30222222222223 )
RUN 20 : 205.8111111111111 / 250 ( %82.32444444444444 )
RUN 21 : 205.61111111111111 / 250 ( %82.24444444444444 )
RUN 22 : 205.7 / 250 ( %82.28 )
RUN 23 : 206.61111111111111 / 250 ( %82.64444444444445 )
RUN 24 : 204.73333333333332 / 250 ( %81.89333333333333 )
RUN 25 : 205.54444444444445 / 250 ( %82.21777777777778 )
_____
AVERAGE : 206.23733333333334 / 250 ( %82.49493333333334 )
STANDARD DEVIATION : 0.9122317227817626 / 250 ( %0.36489268911270506 )
BEST RESULT : 207.85555555555555 / 250 ( %83.14222222222222 )
WORST RESULT : 204.73333333333332 / 250 ( %81.89333333333333 )
MEDIAN : 206.06111111111113 / 250 ( %82.42444444444446 )
**********************
```

3. Problem tanımı için sonuçlar

```
RESULTS FOR PARAMETER SET – 3 :
RUN 1 : 207.56666666666666 / 250 ( %83.02666666666666 )
RUN 2 : 207.62222222222223 / 250 ( %83.0488888888889 )
RUN 3 : 205.77777777777777 / 250 ( %82.31111111111112 )
RUN 4 : 206.64444444444445 / 250 ( %82.65777777777778 )
RUN 5 : 206.87777777777777 / 250 ( %82.75111111111111 )
RUN 6 : 206.53333333333333 / 250 ( %82.61333333333333 )
RUN 7 : 206.7 / 250 ( %82.67999999999999 )
RUN 8 : 204.72222222222223 / 250 ( %81.88888888888889 )
RUN 9 : 206.82222222222222 / 250 ( %82.72888888888889 )
RUN 10 : 206.72222222222223 / 250 ( %82.68888888888888 )
RUN 11 : 207.2111111111111 / 250 ( %82.88444444444444 )
RUN 12 : 205.66666666666666 / 250 ( %82.26666666666667 )
RUN 13 : 206.8111111111111 / 250 ( %82.72444444444444 )
RUN 14 : 206.7111111111111 / 250 ( %82.68444444444444 )
RUN 15 : 206.6 / 250 ( %82.64 )
RUN 16 : 205.83333333333334 / 250 ( %82.33333333333334 )
RUN 17 : 205.62222222222223 / 250 ( %82.2488888888889 )
RUN 18 : 205.77777777777777 / 250 ( %82.31111111111112 )
RUN 19 : 203.4111111111111 / 250 ( %81.36444444444444 )
RUN 20 : 205.7 / 250 ( %82.28 )
RUN 21 : 206.7111111111111 / 250 ( %82.68444444444444 )
RUN 22 : 207.6888888888889 / 250 ( %83.07555555555555 )
RUN 23 : 206.83333333333334 / 250 ( %82.73333333333333 )
RUN 24 : 206.64444444444445 / 250 ( %82.65777777777778 )
RUN 25 : 205.56666666666666 / 250 ( %82.22666666666667 )
_____
AVERAGE : 206.3511111111111 / 250 ( %82.54044444444443 )
STANDARD DEVIATION : 0.9298028464859297 / 250 ( %0.37192113859437187 )
BEST RESULT : 207.6888888888889 / 250 ( %83.07555555555555 )
WORST RESULT : 203.4111111111111 / 250 ( %81.36444444444444 )
MEDIAN : 206.23888888888888 / 250 ( %82.49555555555555 )
**********************
```

4. Problem tanımı için sonuçlar

(SÜREKLİ OPTİMİZASYON HESAP VE SONUÇLARI – BÖLÜM 3)

(DE/RAND/1)

```
 MUTATION METHOD -> DE/RAND/1 -> METRICS

RESULTS FOR PARAMETER SET - 0 :
RUN 1 : 263.0 / 300 ( %87.66666666666667 )
RUN 2 : 270.0 / 300 ( %90.0 )
RUN 3 : 277.0 / 300 ( %92.33333333333333 )
RUN 4 : 274.0 / 300 ( %91.33333333333333 )
RUN 5 : 265.0 / 300 ( %88.33333333333333 )
RUN 6 : 262.0 / 300 ( %87.33333333333333 )
RUN 7 : 264.0 / 300 ( %88.0 )
RUN 8 : 267.0 / 300 ( %89.0 )
RUN 9 : 271.0 / 300 ( %90.33333333333333 )
RUN 10 : 273.0 / 300 ( %91.0 )
RUN 11 : 262.0 / 300 ( %87.33333333333333 )
RUN 12 : 270.0 / 300 ( %90.0 )
RUN 13 : 256.0 / 300 ( %85.33333333333334 )
RUN 14 : 267.0 / 300 ( %89.0 )
RUN 15 : 262.0 / 300 ( %87.33333333333333 )
RUN 16 : 270.0 / 300 ( %90.0 )
RUN 17 : 256.0 / 300 ( %85.33333333333334 )
RUN 18 : 265.0 / 300 ( %88.33333333333333 )
RUN 19 : 262.0 / 300 ( %87.33333333333333 )
RUN 20 : 269.0 / 300 ( %89.66666666666666 )
RUN 21 : 268.0 / 300 ( %89.33333333333333 )
RUN 22 : 264.0 / 300 ( %88.0 )
RUN 23 : 266.0 / 300 ( %88.66666666666667 )
RUN 24 : 266.0 / 300 ( %88.66666666666667 )
RUN 25 : 259.0 / 300 ( %86.33333333333333 )
_____________________
AVERAGE : 265.92 / 300 ( %88.64000000000001 )
STANDARD DEVIATION : 5.114059053237457 / 300 ( %1.7046863510791523 )
BEST RESULT : 277.0 / 300 ( %92.33333333333333 )
WORST RESULT : 256.0 / 300 ( %85.33333333333334 )
MEDIAN : 263.0 / 300 ( %87.66666666666667 )
**********************
```

1. Problem tanımı için sonuçlar

```
RESULTS FOR PARAMETER SET - 1 :
RUN 1 : 265.0 / 300 ( %88.33333333333333 )
RUN 2 : 272.0 / 300 ( %90.66666666666666 )
RUN 3 : 275.0 / 300 ( %91.66666666666666 )
RUN 4 : 255.0 / 300 ( %85.0 )
RUN 5 : 271.0 / 300 ( %90.33333333333333 )
RUN 6 : 260.0 / 300 ( %86.66666666666667 )
RUN 7 : 269.0 / 300 ( %89.66666666666666 )
RUN 8 : 271.0 / 300 ( %90.33333333333333 )
RUN 9 : 269.0 / 300 ( %89.66666666666666 )
RUN 10 : 266.0 / 300 ( %88.66666666666667 )
RUN 11 : 251.0 / 300 ( %83.66666666666667 )
RUN 12 : 268.0 / 300 ( %89.33333333333333 )
RUN 13 : 271.0 / 300 ( %90.33333333333333 )
RUN 14 : 268.0 / 300 ( %89.33333333333333 )
RUN 15 : 257.0 / 300 ( %85.66666666666667 )
RUN 16 : 267.0 / 300 ( %89.0 )
RUN 17 : 269.0 / 300 ( %89.66666666666666 )
RUN 18 : 271.0 / 300 ( %90.33333333333333 )
RUN 19 : 267.0 / 300 ( %89.0 )
RUN 20 : 259.0 / 300 ( %86.33333333333333 )
RUN 21 : 269.0 / 300 ( %89.66666666666666 )
RUN 22 : 266.0 / 300 ( %88.66666666666667 )
RUN 23 : 273.0 / 300 ( %91.0 )
RUN 24 : 269.0 / 300 ( %89.66666666666666 )
RUN 25 : 268.0 / 300 ( %89.33333333333333 )
_____
AVERAGE : 266.64 / 300 ( %88.88 )
STANDARD DEVIATION : 5.754163709871312 / 300 ( %1.918054569957104 )
BEST RESULT : 275.0 / 300 ( %91.66666666666666 )
WORST RESULT : 251.0 / 300 ( %83.66666666666667 )
MEDIAN : 269.5 / 300 ( %89.83333333333333 )
**********
```

2.problem tanımı için sonuçlar

```
RESULTS FOR PARAMETER SET - 2 :
RUN 1 : 254.0 / 300 ( %84.66666666666667 )
RUN 2 : 262.0 / 300 ( %87.33333333333333 )
RUN 3 : 263.0 / 300 ( %87.66666666666667 )
RUN 4 : 263.0 / 300 ( %87.66666666666667 )
RUN 5 : 265.0 / 300 ( %88.33333333333333 )
RUN 6 : 259.0 / 300 ( %86.33333333333333 )
RUN 7 : 257.0 / 300 ( %85.66666666666667 )
RUN 8 : 262.0 / 300 ( %87.33333333333333 )
RUN 9 : 266.0 / 300 ( %88.66666666666667 )
RUN 10 : 270.0 / 300 ( %90.0 )
RUN 11 : 275.0 / 300 ( %91.66666666666666 )
RUN 12 : 266.0 / 300 ( %88.66666666666667 )
RUN 13 : 270.0 / 300 ( %90.0 )
RUN 14 : 262.0 / 300 ( %87.33333333333333 )
RUN 15 : 265.0 / 300 ( %88.33333333333333 )
RUN 16 : 262.0 / 300 ( %87.33333333333333 )
RUN 17 : 271.0 / 300 ( %90.33333333333333 )
RUN 18 : 269.0 / 300 ( %89.66666666666666 )
RUN 19 : 273.0 / 300 ( %91.0 )
RUN 20 : 265.0 / 300 ( %88.33333333333333 )
RUN 21 : 266.0 / 300 ( %88.66666666666667 )
RUN 22 : 258.0 / 300 ( %86.0 )
RUN 23 : 263.0 / 300 ( %87.66666666666667 )
RUN 24 : 267.0 / 300 ( %89.0 )
RUN 25 : 263.0 / 300 ( %87.66666666666667 )
_____________________
AVERAGE : 264.64 / 300 ( %88.21333333333334 )
STANDARD DEVIATION : 4.865223530322115 / 300 ( %1.6217411767740386 )
BEST RESULT : 275.0 / 300 ( %91.66666666666666 )
WORST RESULT : 254.0 / 300 ( %84.66666666666667 )
MEDIAN : 268.0 / 300 ( %89.33333333333333 )
**********************
```

3.problem tanımı için sonuçlar

```
RESULTS FOR PARAMETER SET - 3 :
RUN 1 : 268.0 / 300 ( %89.33333333333333 )
RUN 2 : 263.0 / 300 ( %87.66666666666667 )
RUN 3 : 269.0 / 300 ( %89.66666666666666 )
RUN 4 : 264.0 / 300 ( %88.0 )
RUN 5 : 272.0 / 300 ( %90.66666666666666 )
RUN 6 : 266.0 / 300 ( %88.66666666666667 )
RUN 7 : 277.0 / 300 ( %92.33333333333333 )
RUN 8 : 274.0 / 300 ( %91.33333333333333 )
RUN 9 : 270.0 / 300 ( %90.0 )
RUN 10 : 264.0 / 300 ( %88.0 )
RUN 11 : 272.0 / 300 ( %90.66666666666666 )
RUN 12 : 263.0 / 300 ( %87.66666666666667 )
RUN 13 : 269.0 / 300 ( %89.66666666666666 )
RUN 14 : 257.0 / 300 ( %85.66666666666667 )
RUN 15 : 259.0 / 300 ( %86.33333333333333 )
RUN 16 : 260.0 / 300 ( %86.66666666666667 )
RUN 17 : 267.0 / 300 ( %89.0 )
RUN 18 : 265.0 / 300 ( %88.33333333333333 )
RUN 19 : 274.0 / 300 ( %91.33333333333333 )
RUN 20 : 266.0 / 300 ( %88.66666666666667 )
RUN 21 : 264.0 / 300 ( %88.0 )
RUN 22 : 262.0 / 300 ( %87.33333333333333 )
RUN 23 : 276.0 / 300 ( %92.0 )
RUN 24 : 262.0 / 300 ( %87.33333333333333 )
RUN 25 : 264.0 / 300 ( %88.0 )

_____________________
AVERAGE : 266.68 / 300 ( %88.89333333333333 )
STANDARD DEVIATION : 5.2208811516831135 / 300 ( %1.7402937172277044 )
BEST RESULT : 277.0 / 300 ( %92.33333333333333 )
WORST RESULT : 257.0 / 300 ( %85.66666666666667 )
MEDIAN : 266.0 / 300 ( %88.66666666666667 )
**********************
```

4.problem tanımı için sonuçlar

(DE/RAND/2)

```
 MUTATION METHOD -> DE/RAND/2 -> METRICS

RESULTS FOR PARAMETER SET - 0 :
RUN 1 : 268.0 / 300 ( %89.33333333333333 )
RUN 2 : 270.0 / 300 ( %90.0 )
RUN 3 : 271.0 / 300 ( %90.33333333333333 )
RUN 4 : 270.0 / 300 ( %90.0 )
RUN 5 : 273.0 / 300 ( %91.0 )
RUN 6 : 275.0 / 300 ( %91.66666666666666 )
RUN 7 : 269.0 / 300 ( %89.66666666666666 )
RUN 8 : 272.0 / 300 ( %90.66666666666666 )
RUN 9 : 261.0 / 300 ( %87.0 )
RUN 10 : 270.0 / 300 ( %90.0 )
RUN 11 : 215.18 / 300 ( %71.72666666666667 )
RUN 12 : 274.0 / 300 ( %91.33333333333333 )
RUN 13 : 271.0 / 300 ( %90.33333333333333 )
RUN 14 : 273.0 / 300 ( %91.0 )
RUN 15 : 269.0 / 300 ( %89.66666666666666 )
RUN 16 : 274.0 / 300 ( %91.33333333333333 )
RUN 17 : 267.0 / 300 ( %89.0 )
RUN 18 : 272.0 / 300 ( %90.66666666666666 )
RUN 19 : 273.0 / 300 ( %91.0 )
RUN 20 : 262.0 / 300 ( %87.33333333333333 )
RUN 21 : 277.0 / 300 ( %92.33333333333333 )
RUN 22 : 271.0 / 300 ( %90.33333333333333 )
RUN 23 : 280.0 / 300 ( %93.33333333333333 )
RUN 24 : 267.0 / 300 ( %89.0 )
RUN 25 : 260.0 / 300 ( %86.66666666666667 )
_____________________
AVERAGE : 268.16720000000004 / 300 ( %89.38906666666668 )
STANDARD DEVIATION : 11.720500849366463 / 300 ( %3.9068336164554878 )
BEST RESULT : 280.0 / 300 ( %93.33333333333333 )
WORST RESULT : 215.18 / 300 ( %71.72666666666667 )
MEDIAN : 272.5 / 300 ( %90.83333333333333 )
**********************
```

1.problem tanımı için sonuçlar

```
RESULTS FOR PARAMETER SET - 1 :
RUN 1 : 274.21 / 300 ( %91.40333333333332 )
RUN 2 : 277.0 / 300 ( %92.33333333333333 )
RUN 3 : 273.0 / 300 ( %91.0 )
RUN 4 : 267.0 / 300 ( %89.0 )
RUN 5 : 270.0 / 300 ( %90.0 )
RUN 6 : 269.0 / 300 ( %89.66666666666666 )
RUN 7 : 273.0 / 300 ( %91.0 )
RUN 8 : 269.0 / 300 ( %89.66666666666666 )
RUN 9 : 272.0 / 300 ( %90.66666666666666 )
RUN 10 : 270.0 / 300 ( %90.0 )
RUN 11 : 276.0 / 300 ( %92.0 )
RUN 12 : 273.0 / 300 ( %91.0 )
RUN 13 : 261.0 / 300 ( %87.0 )
RUN 14 : 269.0 / 300 ( %89.66666666666666 )
RUN 15 : 210.35 / 300 ( %70.11666666666666 )
RUN 16 : 270.0 / 300 ( %90.0 )
RUN 17 : 208.48 / 300 ( %69.49333333333333 )
RUN 18 : 271.0 / 300 ( %90.33333333333333 )
RUN 19 : 273.0 / 300 ( %91.0 )
RUN 20 : 272.0 / 300 ( %90.66666666666666 )
RUN 21 : 201.69 / 300 ( %67.23 )
RUN 22 : 273.0 / 300 ( %91.0 )
RUN 23 : 275.0 / 300 ( %91.66666666666666 )
RUN 24 : 277.0 / 300 ( %92.33333333333333 )
RUN 25 : 266.0 / 300 ( %88.66666666666667 )
_____
AVERAGE : 263.62919999999997 / 300 ( %87.87639999999999 )
STANDARD DEVIATION : 21.29237965470276 / 300 ( %7.097459884900919 )
BEST RESULT : 277.0 / 300 ( %92.33333333333333 )
WORST RESULT : 201.69 / 300 ( %67.23 )
MEDIAN : 267.0 / 300 ( %89.0 )
**********
```

2.problem tanımı için sonuçlar

```
RESULTS FOR PARAMETER SET - 2 :
RUN 1 : 271.0 / 300 ( %90.33333333333333 )
RUN 2 : 264.0 / 300 ( %88.0 )
RUN 3 : 265.0 / 300 ( %88.33333333333333 )
RUN 4 : 262.0 / 300 ( %87.33333333333333 )
RUN 5 : 263.0 / 300 ( %87.66666666666667 )
RUN 6 : 277.0 / 300 ( %92.33333333333333 )
RUN 7 : 270.0 / 300 ( %90.0 )
RUN 8 : 274.0 / 300 ( %91.33333333333333 )
RUN 9 : 224.55 / 300 ( %74.85000000000001 )
RUN 10 : 265.8 / 300 ( %88.6 )
RUN 11 : 268.0 / 300 ( %89.33333333333333 )
RUN 12 : 265.0 / 300 ( %88.33333333333333 )
RUN 13 : 271.0 / 300 ( %90.33333333333333 )
RUN 14 : 208.99 / 300 ( %69.66333333333333 )
RUN 15 : 269.0 / 300 ( %89.66666666666666 )
RUN 16 : 274.0 / 300 ( %91.33333333333333 )
RUN 17 : 268.0 / 300 ( %89.33333333333333 )
RUN 18 : 267.0 / 300 ( %89.0 )
RUN 19 : 202.44 / 300 ( %67.47999999999999 )
RUN 20 : 274.0 / 300 ( %91.33333333333333 )
RUN 21 : 212.44 / 300 ( %70.81333333333333 )
RUN 22 : 272.0 / 300 ( %90.66666666666666 )
RUN 23 : 265.0 / 300 ( %88.33333333333333 )
RUN 24 : 268.0 / 300 ( %89.33333333333333 )
RUN 25 : 262.0 / 300 ( %87.33333333333333 )
_____________________
AVERAGE : 259.3288 / 300 ( %86.44293333333334 )
STANDARD DEVIATION : 21.212455835192678 / 300 ( %7.0708186117308935 )
BEST RESULT : 277.0 / 300 ( %92.33333333333333 )
WORST RESULT : 202.44 / 300 ( %67.47999999999999 )
MEDIAN : 268.0 / 300 ( %89.33333333333333 )
**********************
```

3.problem tanımı için sonuçlar

```
RESULTS FOR PARAMETER SET - 3 :
RUN 1 : 276.0 / 300 ( %92.0 )
RUN 2 : 269.0 / 300 ( %89.66666666666666 )
RUN 3 : 276.0 / 300 ( %92.0 )
RUN 4 : 268.0 / 300 ( %89.33333333333333 )
RUN 5 : 269.0 / 300 ( %89.66666666666666 )
RUN 6 : 271.0 / 300 ( %90.33333333333333 )
RUN 7 : 271.0 / 300 ( %90.33333333333333 )
RUN 8 : 266.0 / 300 ( %88.66666666666667 )
RUN 9 : 270.0 / 300 ( %90.0 )
RUN 10 : 276.0 / 300 ( %92.0 )
RUN 11 : 202.43 / 300 ( %67.47666666666667 )
RUN 12 : 271.0 / 300 ( %90.33333333333333 )
RUN 13 : 274.0 / 300 ( %91.33333333333333 )
RUN 14 : 272.0 / 300 ( %90.66666666666666 )
RUN 15 : 266.0 / 300 ( %88.66666666666667 )
RUN 16 : 215.09 / 300 ( %71.69666666666666 )
RUN 17 : 208.49 / 300 ( %69.49666666666667 )
RUN 18 : 264.0 / 300 ( %88.0 )
RUN 19 : 273.0 / 300 ( %91.0 )
RUN 20 : 273.0 / 300 ( %91.0 )
RUN 21 : 202.76 / 300 ( %67.58666666666666 )
RUN 22 : 259.0 / 300 ( %86.33333333333333 )
RUN 23 : 274.0 / 300 ( %91.33333333333333 )
RUN 24 : 267.0 / 300 ( %89.0 )
RUN 25 : 271.0 / 300 ( %90.33333333333333 )
_____
AVERAGE : 260.19079999999997 / 300 ( %86.73026666666665 )
STANDARD DEVIATION : 23.537205088115282 / 300 ( %7.8457350293717605 )
BEST RESULT : 276.0 / 300 ( %92.0 )
WORST RESULT : 202.43 / 300 ( %67.47666666666667 )
MEDIAN : 272.5 / 300 ( %90.83333333333333 )
**********************
```

4.problem tanımı için sonuçlar

(DE/BEST/1)

```
 MUTATION METHOD -> DE/BEST/1 -> METRICS

RESULTS FOR PARAMETER SET - 0 :
RUN 1 : 212.0 / 300 ( %70.66666666666667 )
RUN 2 : 211.0 / 300 ( %70.33333333333334 )
RUN 3 : 210.0 / 300 ( %70.0 )
RUN 4 : 202.0 / 300 ( %67.33333333333333 )
RUN 5 : 205.0 / 300 ( %68.33333333333333 )
RUN 6 : 212.0 / 300 ( %70.66666666666667 )
RUN 7 : 212.0 / 300 ( %70.66666666666667 )
RUN 8 : 214.0 / 300 ( %71.33333333333334 )
RUN 9 : 215.0 / 300 ( %71.66666666666667 )
RUN 10 : 208.0 / 300 ( %69.33333333333334 )
RUN 11 : 206.0 / 300 ( %68.66666666666667 )
RUN 12 : 205.0 / 300 ( %68.33333333333333 )
RUN 13 : 210.0 / 300 ( %70.0 )
RUN 14 : 208.0 / 300 ( %69.33333333333334 )
RUN 15 : 214.0 / 300 ( %71.33333333333334 )
RUN 16 : 215.0 / 300 ( %71.66666666666667 )
RUN 17 : 205.0 / 300 ( %68.33333333333333 )
RUN 18 : 204.0 / 300 ( %68.0 )
RUN 19 : 206.0 / 300 ( %68.66666666666667 )
RUN 20 : 208.0 / 300 ( %69.33333333333334 )
RUN 21 : 211.0 / 300 ( %70.33333333333334 )
RUN 22 : 204.0 / 300 ( %68.0 )
RUN 23 : 209.0 / 300 ( %69.66666666666667 )
RUN 24 : 207.0 / 300 ( %69.0 )
RUN 25 : 210.0 / 300 ( %70.0 )
_____________________
AVERAGE : 208.92 / 300 ( %69.63999999999999 )
STANDARD DEVIATION : 3.6432952117554236 / 300 ( %1.2144317372518079 )
BEST RESULT : 215.0 / 300 ( %71.66666666666667 )
WORST RESULT : 202.0 / 300 ( %67.33333333333333 )
MEDIAN : 207.5 / 300 ( %69.16666666666667 )
**********************
```

1.problem tanımı için sonuçlar

```
RESULTS FOR PARAMETER SET - 1 :
RUN 1 : 217.0 / 300 ( %72.33333333333334 )
RUN 2 : 202.0 / 300 ( %67.33333333333333 )
RUN 3 : 201.0 / 300 ( %67.0 )
RUN 4 : 210.0 / 300 ( %70.0 )
RUN 5 : 217.0 / 300 ( %72.33333333333334 )
RUN 6 : 205.0 / 300 ( %68.33333333333333 )
RUN 7 : 208.0 / 300 ( %69.33333333333334 )
RUN 8 : 212.0 / 300 ( %70.66666666666667 )
RUN 9 : 209.0 / 300 ( %69.66666666666667 )
RUN 10 : 214.0 / 300 ( %71.33333333333334 )
RUN 11 : 208.0 / 300 ( %69.33333333333334 )
RUN 12 : 210.0 / 300 ( %70.0 )
RUN 13 : 210.0 / 300 ( %70.0 )
RUN 14 : 208.0 / 300 ( %69.33333333333334 )
RUN 15 : 208.0 / 300 ( %69.33333333333334 )
RUN 16 : 210.0 / 300 ( %70.0 )
RUN 17 : 216.0 / 300 ( %72.0 )
RUN 18 : 218.0 / 300 ( %72.66666666666667 )
RUN 19 : 217.0 / 300 ( %72.33333333333334 )
RUN 20 : 214.0 / 300 ( %71.33333333333334 )
RUN 21 : 213.0 / 300 ( %71.0 )
RUN 22 : 217.0 / 300 ( %72.33333333333334 )
RUN 23 : 216.0 / 300 ( %72.0 )
RUN 24 : 211.0 / 300 ( %70.33333333333334 )
RUN 25 : 213.0 / 300 ( %71.0 )
_____
AVERAGE : 211.36 / 300 ( %70.45333333333333 )
STANDARD DEVIATION : 4.5946055325783925 / 300 ( %1.5315351775261308 )
BEST RESULT : 218.0 / 300 ( %72.66666666666667 )
WORST RESULT : 201.0 / 300 ( %67.0 )
MEDIAN : 210.0 / 300 ( %70.0 )
**********
```

2.problem tanımı için sonuçlar

```
RESULTS FOR PARAMETER SET - 2 :
RUN 1 : 217.0 / 300 ( %72.33333333333334 )
RUN 2 : 216.0 / 300 ( %72.0 )
RUN 3 : 206.0 / 300 ( %68.66666666666667 )
RUN 4 : 205.0 / 300 ( %68.33333333333333 )
RUN 5 : 216.0 / 300 ( %72.0 )
RUN 6 : 208.0 / 300 ( %69.33333333333334 )
RUN 7 : 214.0 / 300 ( %71.33333333333334 )
RUN 8 : 208.0 / 300 ( %69.33333333333334 )
RUN 9 : 205.0 / 300 ( %68.33333333333333 )
RUN 10 : 207.0 / 300 ( %69.0 )
RUN 11 : 203.0 / 300 ( %67.66666666666666 )
RUN 12 : 210.0 / 300 ( %70.0 )
RUN 13 : 207.0 / 300 ( %69.0 )
RUN 14 : 218.0 / 300 ( %72.66666666666667 )
RUN 15 : 216.0 / 300 ( %72.0 )
RUN 16 : 218.0 / 300 ( %72.66666666666667 )
RUN 17 : 205.0 / 300 ( %68.33333333333333 )
RUN 18 : 206.0 / 300 ( %68.66666666666667 )
RUN 19 : 207.0 / 300 ( %69.0 )
RUN 20 : 208.0 / 300 ( %69.33333333333334 )
RUN 21 : 211.0 / 300 ( %70.33333333333334 )
RUN 22 : 203.0 / 300 ( %67.66666666666666 )
RUN 23 : 212.0 / 300 ( %70.66666666666667 )
RUN 24 : 210.0 / 300 ( %70.0 )
RUN 25 : 215.0 / 300 ( %71.66666666666667 )
_____
AVERAGE : 210.04 / 300 ( %70.01333333333332 )
STANDARD DEVIATION : 4.795664708880303 / 300 ( %1.5985549029601012 )
BEST RESULT : 218.0 / 300 ( %72.66666666666667 )
WORST RESULT : 203.0 / 300 ( %67.66666666666666 )
MEDIAN : 208.5 / 300 ( %69.5 )
**********
```

3.problem tanımı için sonuçlar

```
RESULTS FOR PARAMETER SET - 3 :
RUN 1 : 217.0 / 300 ( %72.33333333333334 )
RUN 2 : 216.0 / 300 ( %72.0 )
RUN 3 : 214.0 / 300 ( %71.33333333333334 )
RUN 4 : 212.0 / 300 ( %70.66666666666667 )
RUN 5 : 202.0 / 300 ( %67.33333333333333 )
RUN 6 : 206.0 / 300 ( %68.66666666666667 )
RUN 7 : 217.0 / 300 ( %72.33333333333334 )
RUN 8 : 212.0 / 300 ( %70.66666666666667 )
RUN 9 : 208.0 / 300 ( %69.33333333333334 )
RUN 10 : 214.0 / 300 ( %71.33333333333334 )
RUN 11 : 205.0 / 300 ( %68.33333333333333 )
RUN 12 : 204.0 / 300 ( %68.0 )
RUN 13 : 210.0 / 300 ( %70.0 )
RUN 14 : 210.0 / 300 ( %70.0 )
RUN 15 : 210.0 / 300 ( %70.0 )
RUN 16 : 201.0 / 300 ( %67.0 )
RUN 17 : 207.0 / 300 ( %69.0 )
RUN 18 : 206.0 / 300 ( %68.66666666666667 )
RUN 19 : 209.0 / 300 ( %69.66666666666667 )
RUN 20 : 206.0 / 300 ( %68.66666666666667 )
RUN 21 : 213.0 / 300 ( %71.0 )
RUN 22 : 209.0 / 300 ( %69.66666666666667 )
RUN 23 : 212.0 / 300 ( %70.66666666666667 )
RUN 24 : 212.0 / 300 ( %70.66666666666667 )
RUN 25 : 210.0 / 300 ( %70.0 )
_____________________
AVERAGE : 209.68 / 300 ( %69.89333333333335 )
STANDARD DEVIATION : 4.277569403294351 / 300 ( %1.4258564677647838 )
BEST RESULT : 217.0 / 300 ( %72.33333333333334 )
WORST RESULT : 201.0 / 300 ( %67.0 )
MEDIAN : 207.0 / 300 ( %69.0 )
**********************
```

4.problem tanımı için sonuçlar

(DE/BEST/2)

```
 MUTATION METHOD -> DE/BEST/2 -> METRICS

RESULTS FOR PARAMETER SET - 0 :
RUN 1 : 216.0 / 300 ( %72.0 )
RUN 2 : 215.0 / 300 ( %71.66666666666667 )
RUN 3 : 210.0 / 300 ( %70.0 )
RUN 4 : 215.0 / 300 ( %71.66666666666667 )
RUN 5 : 212.0 / 300 ( %70.66666666666667 )
RUN 6 : 219.0 / 300 ( %73.0 )
RUN 7 : 206.0 / 300 ( %68.66666666666667 )
RUN 8 : 206.0 / 300 ( %68.66666666666667 )
RUN 9 : 218.0 / 300 ( %72.66666666666667 )
RUN 10 : 212.0 / 300 ( %70.66666666666667 )
RUN 11 : 206.0 / 300 ( %68.66666666666667 )
RUN 12 : 224.0 / 300 ( %74.66666666666667 )
RUN 13 : 220.0 / 300 ( %73.33333333333333 )
RUN 14 : 215.0 / 300 ( %71.66666666666667 )
RUN 15 : 208.0 / 300 ( %69.33333333333334 )
RUN 16 : 217.0 / 300 ( %72.33333333333334 )
RUN 17 : 221.0 / 300 ( %73.66666666666667 )
RUN 18 : 217.0 / 300 ( %72.33333333333334 )
RUN 19 : 206.0 / 300 ( %68.66666666666667 )
RUN 20 : 215.0 / 300 ( %71.66666666666667 )
RUN 21 : 221.0 / 300 ( %73.66666666666667 )
RUN 22 : 205.0 / 300 ( %68.33333333333333 )
RUN 23 : 212.0 / 300 ( %70.66666666666667 )
RUN 24 : 211.0 / 300 ( %70.33333333333334 )
RUN 25 : 213.0 / 300 ( %71.0 )
_____________________
AVERAGE : 213.6 / 300 ( %71.2 )
STANDARD DEVIATION : 5.321653878260029 / 300 ( %1.773884626086676 )
BEST RESULT : 224.0 / 300 ( %74.66666666666667 )
WORST RESULT : 205.0 / 300 ( %68.33333333333333 )
MEDIAN : 222.0 / 300 ( %74.0 )
**********************
```

1.problem tanımı için sonuçlar

```
RESULTS FOR PARAMETER SET - 1 :
RUN 1 : 209.0 / 300 ( %69.66666666666667 )
RUN 2 : 216.0 / 300 ( %72.0 )
RUN 3 : 211.0 / 300 ( %70.33333333333334 )
RUN 4 : 209.0 / 300 ( %69.66666666666667 )
RUN 5 : 212.0 / 300 ( %70.66666666666667 )
RUN 6 : 212.0 / 300 ( %70.66666666666667 )
RUN 7 : 220.0 / 300 ( %73.33333333333333 )
RUN 8 : 223.0 / 300 ( %74.33333333333333 )
RUN 9 : 214.0 / 300 ( %71.33333333333334 )
RUN 10 : 212.0 / 300 ( %70.66666666666667 )
RUN 11 : 212.0 / 300 ( %70.66666666666667 )
RUN 12 : 220.0 / 300 ( %73.33333333333333 )
RUN 13 : 209.0 / 300 ( %69.66666666666667 )
RUN 14 : 212.0 / 300 ( %70.66666666666667 )
RUN 15 : 212.0 / 300 ( %70.66666666666667 )
RUN 16 : 211.0 / 300 ( %70.33333333333334 )
RUN 17 : 211.0 / 300 ( %70.33333333333334 )
RUN 18 : 215.0 / 300 ( %71.66666666666667 )
RUN 19 : 215.0 / 300 ( %71.66666666666667 )
RUN 20 : 210.0 / 300 ( %70.0 )
RUN 21 : 210.0 / 300 ( %70.0 )
RUN 22 : 217.0 / 300 ( %72.33333333333334 )
RUN 23 : 220.0 / 300 ( %73.33333333333333 )
RUN 24 : 214.0 / 300 ( %71.33333333333334 )
RUN 25 : 222.0 / 300 ( %74.0 )
_____________________
AVERAGE : 213.92 / 300 ( %71.30666666666666 )
STANDARD DEVIATION : 4.127178212774437 / 300 ( %1.3757260709248125 )
BEST RESULT : 223.0 / 300 ( %74.33333333333333 )
WORST RESULT : 209.0 / 300 ( %69.66666666666667 )
MEDIAN : 214.5 / 300 ( %71.5 )
**********************
```

2.problem tanımı için sonuçlar

```
RESULTS FOR PARAMETER SET - 2 :
RUN 1 : 208.0 / 300 ( %69.33333333333334 )
RUN 2 : 216.0 / 300 ( %72.0 )
RUN 3 : 208.0 / 300 ( %69.33333333333334 )
RUN 4 : 211.0 / 300 ( %70.33333333333334 )
RUN 5 : 205.0 / 300 ( %68.33333333333333 )
RUN 6 : 203.0 / 300 ( %67.66666666666666 )
RUN 7 : 208.0 / 300 ( %69.33333333333334 )
RUN 8 : 218.0 / 300 ( %72.66666666666667 )
RUN 9 : 209.0 / 300 ( %69.66666666666667 )
RUN 10 : 206.0 / 300 ( %68.66666666666667 )
RUN 11 : 207.0 / 300 ( %69.0 )
RUN 12 : 213.0 / 300 ( %71.0 )
RUN 13 : 216.0 / 300 ( %72.0 )
RUN 14 : 205.0 / 300 ( %68.33333333333333 )
RUN 15 : 223.0 / 300 ( %74.33333333333333 )
RUN 16 : 209.0 / 300 ( %69.66666666666667 )
RUN 17 : 221.0 / 300 ( %73.66666666666667 )
RUN 18 : 210.0 / 300 ( %70.0 )
RUN 19 : 219.0 / 300 ( %73.0 )
RUN 20 : 211.0 / 300 ( %70.33333333333334 )
RUN 21 : 211.0 / 300 ( %70.33333333333334 )
RUN 22 : 217.0 / 300 ( %72.33333333333334 )
RUN 23 : 212.0 / 300 ( %70.66666666666667 )
RUN 24 : 216.0 / 300 ( %72.0 )
RUN 25 : 212.0 / 300 ( %70.66666666666667 )
_____
AVERAGE : 211.76 / 300 ( %70.58666666666666 )
STANDARD DEVIATION : 5.186752355761743 / 300 ( %1.7289174519205812 )
BEST RESULT : 223.0 / 300 ( %74.33333333333333 )
WORST RESULT : 203.0 / 300 ( %67.66666666666666 )
MEDIAN : 214.5 / 300 ( %71.5 )
**********
```

3.problem tanımı için sonuçlar

```
RESULTS FOR PARAMETER SET - 3 :
RUN 1 : 217.0 / 300 ( %72.33333333333334 )
RUN 2 : 206.0 / 300 ( %68.66666666666667 )
RUN 3 : 211.0 / 300 ( %70.33333333333334 )
RUN 4 : 211.0 / 300 ( %70.33333333333334 )
RUN 5 : 217.0 / 300 ( %72.33333333333334 )
RUN 6 : 221.0 / 300 ( %73.66666666666667 )
RUN 7 : 207.0 / 300 ( %69.0 )
RUN 8 : 211.0 / 300 ( %70.33333333333334 )
RUN 9 : 224.0 / 300 ( %74.66666666666667 )
RUN 10 : 204.0 / 300 ( %68.0 )
RUN 11 : 211.0 / 300 ( %70.33333333333334 )
RUN 12 : 211.0 / 300 ( %70.33333333333334 )
RUN 13 : 211.0 / 300 ( %70.33333333333334 )
RUN 14 : 208.0 / 300 ( %69.33333333333334 )
RUN 15 : 209.0 / 300 ( %69.66666666666667 )
RUN 16 : 216.0 / 300 ( %72.0 )
RUN 17 : 211.0 / 300 ( %70.33333333333334 )
RUN 18 : 206.0 / 300 ( %68.66666666666667 )
RUN 19 : 214.0 / 300 ( %71.33333333333334 )
RUN 20 : 217.0 / 300 ( %72.33333333333334 )
RUN 21 : 215.0 / 300 ( %71.66666666666667 )
RUN 22 : 210.0 / 300 ( %70.0 )
RUN 23 : 211.0 / 300 ( %70.33333333333334 )
RUN 24 : 211.0 / 300 ( %70.33333333333334 )
RUN 25 : 209.0 / 300 ( %69.66666666666667 )
_____________________
AVERAGE : 211.96 / 300 ( %70.65333333333334 )
STANDARD DEVIATION : 4.634479474547277 / 300 ( %1.544826491515759 )
BEST RESULT : 224.0 / 300 ( %74.66666666666667 )
WORST RESULT : 204.0 / 300 ( %68.0 )
MEDIAN : 211.0 / 300 ( %70.33333333333334 )
**********************
```

4.problem tanımı için sonuçlar

(DE/CURRENT_TO_BEST/1)

```
 MUTATION METHOD -> DE/CURRENT_TO_BEST/1 -> METRICS

RESULTS FOR PARAMETER SET - 0 :
RUN 1 : 211.0 / 300 ( %70.33333333333334 )
RUN 2 : 209.0 / 300 ( %69.66666666666667 )
RUN 3 : 196.0 / 300 ( %65.33333333333333 )
RUN 4 : 208.0 / 300 ( %69.33333333333334 )
RUN 5 : 212.0 / 300 ( %70.66666666666667 )
RUN 6 : 208.0 / 300 ( %69.33333333333334 )
RUN 7 : 204.0 / 300 ( %68.0 )
RUN 8 : 212.0 / 300 ( %70.66666666666667 )
RUN 9 : 209.0 / 300 ( %69.66666666666667 )
RUN 10 : 206.0 / 300 ( %68.66666666666667 )
RUN 11 : 205.0 / 300 ( %68.33333333333333 )
RUN 12 : 209.0 / 300 ( %69.66666666666667 )
RUN 13 : 213.0 / 300 ( %71.0 )
RUN 14 : 206.0 / 300 ( %68.66666666666667 )
RUN 15 : 213.0 / 300 ( %71.0 )
RUN 16 : 212.0 / 300 ( %70.66666666666667 )
RUN 17 : 208.0 / 300 ( %69.33333333333334 )
RUN 18 : 214.0 / 300 ( %71.33333333333334 )
RUN 19 : 209.0 / 300 ( %69.66666666666667 )
RUN 20 : 213.0 / 300 ( %71.0 )
RUN 21 : 208.0 / 300 ( %69.33333333333334 )
RUN 22 : 217.0 / 300 ( %72.33333333333334 )
RUN 23 : 212.0 / 300 ( %70.66666666666667 )
RUN 24 : 211.0 / 300 ( %70.33333333333334 )
RUN 25 : 206.0 / 300 ( %68.66666666666667 )
_____________________
AVERAGE : 209.24 / 300 ( %69.74666666666667 )
STANDARD DEVIATION : 4.101511916354747 / 300 ( %1.3671706387849158 )
BEST RESULT : 217.0 / 300 ( %72.33333333333334 )
WORST RESULT : 196.0 / 300 ( %65.33333333333333 )
MEDIAN : 211.0 / 300 ( %70.33333333333334 )
**********************
```

1.problem tanımı için sonuçlar

```
RESULTS FOR PARAMETER SET - 1 :
RUN 1 : 214.0 / 300 ( %71.33333333333334 )
RUN 2 : 211.0 / 300 ( %70.33333333333334 )
RUN 3 : 204.0 / 300 ( %68.0 )
RUN 4 : 213.0 / 300 ( %71.0 )
RUN 5 : 214.0 / 300 ( %71.33333333333334 )
RUN 6 : 208.0 / 300 ( %69.33333333333334 )
RUN 7 : 215.0 / 300 ( %71.66666666666667 )
RUN 8 : 221.0 / 300 ( %73.66666666666667 )
RUN 9 : 213.0 / 300 ( %71.0 )
RUN 10 : 207.0 / 300 ( %69.0 )
RUN 11 : 205.0 / 300 ( %68.33333333333333 )
RUN 12 : 211.0 / 300 ( %70.33333333333334 )
RUN 13 : 206.0 / 300 ( %68.66666666666667 )
RUN 14 : 206.0 / 300 ( %68.66666666666667 )
RUN 15 : 206.0 / 300 ( %68.66666666666667 )
RUN 16 : 215.0 / 300 ( %71.66666666666667 )
RUN 17 : 210.0 / 300 ( %70.0 )
RUN 18 : 215.0 / 300 ( %71.66666666666667 )
RUN 19 : 211.0 / 300 ( %70.33333333333334 )
RUN 20 : 209.0 / 300 ( %69.66666666666667 )
RUN 21 : 203.0 / 300 ( %67.66666666666666 )
RUN 22 : 210.0 / 300 ( %70.0 )
RUN 23 : 211.0 / 300 ( %70.33333333333334 )
RUN 24 : 208.0 / 300 ( %69.33333333333334 )
RUN 25 : 213.0 / 300 ( %71.0 )
_____________________
AVERAGE : 210.36 / 300 ( %70.12 )
STANDARD DEVIATION : 4.155767077207286 / 300 ( %1.3852556924024286 )
BEST RESULT : 221.0 / 300 ( %73.66666666666667 )
WORST RESULT : 203.0 / 300 ( %67.66666666666666 )
MEDIAN : 208.5 / 300 ( %69.5 )
**********************
```

2.problem tanımı için sonuçlar

```
RESULTS FOR PARAMETER SET - 2 :
RUN 1 : 212.0 / 300 ( %70.66666666666667 )
RUN 2 : 210.0 / 300 ( %70.0 )
RUN 3 : 211.0 / 300 ( %70.33333333333334 )
RUN 4 : 206.0 / 300 ( %68.66666666666667 )
RUN 5 : 213.0 / 300 ( %71.0 )
RUN 6 : 219.0 / 300 ( %73.0 )
RUN 7 : 210.0 / 300 ( %70.0 )
RUN 8 : 204.0 / 300 ( %68.0 )
RUN 9 : 210.0 / 300 ( %70.0 )
RUN 10 : 212.0 / 300 ( %70.66666666666667 )
RUN 11 : 210.0 / 300 ( %70.0 )
RUN 12 : 208.0 / 300 ( %69.33333333333334 )
RUN 13 : 213.0 / 300 ( %71.0 )
RUN 14 : 204.0 / 300 ( %68.0 )
RUN 15 : 210.0 / 300 ( %70.0 )
RUN 16 : 210.0 / 300 ( %70.0 )
RUN 17 : 214.0 / 300 ( %71.33333333333334 )
RUN 18 : 210.0 / 300 ( %70.0 )
RUN 19 : 216.0 / 300 ( %72.0 )
RUN 20 : 215.0 / 300 ( %71.66666666666667 )
RUN 21 : 209.0 / 300 ( %69.66666666666667 )
RUN 22 : 215.0 / 300 ( %71.66666666666667 )
RUN 23 : 214.0 / 300 ( %71.33333333333334 )
RUN 24 : 212.0 / 300 ( %70.66666666666667 )
RUN 25 : 208.0 / 300 ( %69.33333333333334 )
_____
AVERAGE : 211.0 / 300 ( %70.33333333333334 )
STANDARD DEVIATION : 3.4756294393965534 / 300 ( %1.1585431464655178 )
BEST RESULT : 219.0 / 300 ( %73.0 )
WORST RESULT : 204.0 / 300 ( %68.0 )
MEDIAN : 210.5 / 300 ( %70.16666666666667 )
**********
```

3.problem tanımı için sonuçlar

```
RESULTS FOR PARAMETER SET - 3 :
RUN 1 : 203.0 / 300 ( %67.66666666666666 )
RUN 2 : 212.0 / 300 ( %70.66666666666667 )
RUN 3 : 204.0 / 300 ( %68.0 )
RUN 4 : 212.0 / 300 ( %70.66666666666667 )
RUN 5 : 205.0 / 300 ( %68.33333333333333 )
RUN 6 : 216.0 / 300 ( %72.0 )
RUN 7 : 219.0 / 300 ( %73.0 )
RUN 8 : 211.0 / 300 ( %70.33333333333334 )
RUN 9 : 218.0 / 300 ( %72.66666666666667 )
RUN 10 : 213.0 / 300 ( %71.0 )
RUN 11 : 209.0 / 300 ( %69.66666666666667 )
RUN 12 : 208.0 / 300 ( %69.33333333333334 )
RUN 13 : 210.0 / 300 ( %70.0 )
RUN 14 : 204.0 / 300 ( %68.0 )
RUN 15 : 204.0 / 300 ( %68.0 )
RUN 16 : 206.0 / 300 ( %68.66666666666667 )
RUN 17 : 205.0 / 300 ( %68.33333333333333 )
RUN 18 : 209.0 / 300 ( %69.66666666666667 )
RUN 19 : 211.0 / 300 ( %70.33333333333334 )
RUN 20 : 210.0 / 300 ( %70.0 )
RUN 21 : 214.0 / 300 ( %71.33333333333334 )
RUN 22 : 204.0 / 300 ( %68.0 )
RUN 23 : 208.0 / 300 ( %69.33333333333334 )
RUN 24 : 212.0 / 300 ( %70.66666666666667 )
RUN 25 : 203.0 / 300 ( %67.66666666666666 )
_____________________
AVERAGE : 209.2 / 300 ( %69.73333333333332 )
STANDARD DEVIATION : 4.569463863518346 / 300 ( %1.5231546211727822 )
BEST RESULT : 219.0 / 300 ( %73.0 )
WORST RESULT : 203.0 / 300 ( %67.66666666666666 )
MEDIAN : 209.0 / 300 ( %69.66666666666667 )
**********************
```

4.problem tanımı için sonuçlar

5.Sonuç Tabloları

Genetik Algoritmada Kullanılan Parametreler :

- Popülasyon Boyutu : 100
- Jenerasyon Sayısı : 2000
- Elitizm Oranı : %40
- Yaşlı Oranı : %10
- Çocuk Oranı : %89
- Rasgele Birey Oranı : %1
- Çaprazlama Oranı : %90
- Mutasyon Oranı : %10
- Farklı Problem Tanımı Sayısı : 4
- Her Problemi Çözüm Sayısı : 25

	Parametre seti 1	Parametre seti 2	Parametre seti 3	parametre seti 4
Run 1	**%82.595**	**%82.644**	**%82.217**	**%82.026**
Run 2	**%82.280**	**%82.657**	**%82.360**	**%83.048**
Run 3	**%83.040**	**%82.626**	**%82.591**	**%82.311**
Run 4	**%82.960**	**%82.444**	**%82.324**	**%82.657**
Run 5	**%82.755**	**%82.755**	**%82.697**	**%82.751**
Run 6	**%82.653**	**%82.231**	**%83.040**	**%82.613**
Run 7	**%82.288**	**%81.826**	**%82.720**	**%82.679**
Run 8	**%81.493**	**%82.688**	**%82.222**	**%81.888**
Run 9	**%82.266**	**%81.422**	**%83.084**	**%82.728**
Run 10	**%83.080**	**%83.066**	**%83.142**	**%82.688**
Run 11	**%81.795**	**%81.893**	**%81.897**	**%82.884**
Run 12	**%82.711**	**%82.702**	**%82.199**	**%82.266**
Run 13	**%83.031**	**%82.186**	**%82.648**	**%82.724**
Run 14	**%82.631**	**%82.666**	**%82.208**	**%82.684**
Run 15	**%82.995**	**%82.320**	**%82.257**	**%82.640**
Run 16	**%82.631**	**%83.106**	**%82.693**	**%82.333**
Run 17	**%82.640**	**%82.302**	**%83.080**	**%82.248**
Run 18	**%81.884**	**%82.671**	**%83.080**	**%82.311**
Run 19	**%82.333**	**%81.746**	**%82.302**	**%81.364**
Run 20	**%82.711**	**%82.595**	**%82.324**	**%82.280**
Run 21	**%81.933**	**%82.640**	**%82.244**	**%82.684**
Run 22	**%82.626**	**%82.666**	**%82.280**	**%83.075**
Run 23	**%81.937**	**%83.084**	**%82.644**	**%82.733**
Run 24	**%82.640**	**%82.635**	**%81.893**	**%82.657**
Run 25	**%83.075**	**%82.217**	**%82.217**	**%82.226**
Ortalama	**%82.519**	**%82.464**	**%82.494**	**%82.540**
Std. Sapma	**%0.432**	**%0.413**	**%0.364**	**%0.371**
En İyi	**%83.080**	**%83.106**	**%83.142**	**%83.075**
En Kötü	**%81.493**	**%81.422**	**%81.893**	**%81.364**
Medyan	**%82.871**	**%82.444**	**%82.424**	**%82.495**

DE/rand/1 Diferansiyel Gelişim Algoritmasında Kullanılan Parametreler :

- Popülasyon Boyutu : 100
- Jenerasyon Sayısı : 2000
- F : 0.1;
- Çaprazlama/Mutasyon oranı : %90

	Parametre seti 1	Parametre seti 2	Parametre seti 3	parametre seti 4
Run 1	**%87.6**	**%88.3**	**%84.6**	**%89.3**
Run 2	**%90.0**	**%90.6**	**%87.3**	**%87.6**
Run 3	**%92.3**	**%91.6**	**%87.6**	**%89.6**
Run 4	**%91.3**	**%85.0**	**%87.6**	**%88.0**
Run 5	**%88.3**	**%90.3**	**%88.3**	**%90.6**
Run 6	**%87.3**	**%86.6**	**%86.3**	**%88.6**
Run 7	**%88.0**	**%89.6**	**%85.6**	**%92.3**
Run 8	**%89.0**	**%90.3**	**%87.3**	**%91.3**
Run 9	**%90.3**	**%89.6**	**%88.6**	**%90.0**
Run 10	**%91.0**	**%88.6**	**%90.0**	**%88.0**
Run 11	**%87.3**	**%83.6**	**%91.6**	**%90.6**
Run 12	**%90.0**	**%89.3**	**%88.6**	**%87.6**
Run 13	**%85.3**	**%90.3**	**%90.0**	**%89.6**
Run 14	**%89.0**	**%89.3**	**%87.3**	**%85.6**
Run 15	**%87.3**	**%85.6**	**%88.3**	**%86.3**
Run 16	**%90.0**	**%89.0**	**%87.3**	**%86.6**
Run 17	**%85.3**	**%89.6**	**%90.3**	**%89.0**
Run 18	**%88.3**	**%90.3**	**%89.6**	**%88.3**
Run 19	**%87.3**	**%89.0**	**%91.0**	**%91.3**
Run 20	**%89.6**	**%86.3**	**%88.3**	**%88.6**
Run 21	**%89.3**	**%89.6**	**%88.6**	**%88.0**
Run 22	**%88.0**	**%88.6**	**%86.0**	**%87.3**
Run 23	**%88.6**	**%91.0**	**%87.6**	**%92.0**
Run 24	**%88.6**	**%89.6**	**%89.0**	**%87.3**
Run 25	**%86.3**	**%89.3**	**%87.6**	**%88.0**
Ortalama	**%88.640**	**%88.880**	**%88.213**	**%88.893**
Std. Sapma	**%1.704**	**%1.918**	**%1.621**	**%1.740**
En İyi	**%92.3**	**%91.6**	**%91.6**	**%92.3**
En Kötü	**%85.3**	**%83.6**	**%84.6**	**%85.6**
Medyan	**%87.60**	**%89.83**	**%89.30**	**%88.66**

DE/rand/2 Diferansiyel Gelişim Algoritmasında Kullanılan Parametreler :

- Popülasyon Boyutu : 100
- Jenerasyon Sayısı : 2000
- F : 0.1;
- Çaprazlama/Mutasyon oranı : %90

	Parametre seti 1	Parametre seti 2	Parametre seti 3	parametre seti 4
Run 1	**%89.3**	**%91.403**	**%90.3**	**%92.0**
Run 2	**%90.0**	**%92.3**	**%88.0**	**%89.6**
Run 3	**%90.3**	**%91.0**	**%88.3**	**%92.0**
Run 4	**%90.0**	**%89.0**	**%87.3**	**%89.3**
Run 5	**%91.0**	**%90.0**	**%87.6**	**%89.6**
Run 6	**%91.6**	**%89.6**	**%92.3**	**%90.3**
Run 7	**%89.6**	**%91.0**	**%90.0**	**%90.3**
Run 8	**%90.6**	**%89.6**	**%91.3**	**%88.6**
Run 9	**%87.0**	**%90.6**	**%74.85**	**%90.0**
Run 10	**%90.0**	**%90.0**	**%88.6**	**%92.0**
Run 11	**%71.72**	**%92.0**	**%89.3**	**%67.47**
Run 12	**%91.3**	**%91.0**	**%88.3**	**%90.3**
Run 13	**%90.3**	**%87.0**	**%90.3**	**%91.3**
Run 14	**%91.0**	**%89.6**	**%69.66**	**%90.6**
Run 15	**%89.6**	**%70.11**	**%89.66**	**%88.6**
Run 16	**%91.3**	**%90.0**	**%91.33**	**%71.69**
Run 17	**%89.0**	**%69.493**	**%89.33**	**%69.49**
Run 18	**%90.6**	**%90.3**	**%89.0**	**%88.0**
Run 19	**%91.0**	**%91.0**	**%67.47**	**%91.0**
Run 20	**%87.3**	**%90.6**	**%91.3**	**%91.0**
Run 21	**%92.3**	**%67.23**	**%70.81**	**%67.58**
Run 22	**%90.3**	**%91.0**	**%90.6**	**%86.3**
Run 23	**%93.3**	**%91.6**	**%88.3**	**%91.3**
Run 24	**%89.0**	**%92.3**	**%89.3**	**%89.0**
Run 25	**%86.6**	**%88.6**	**%87.3**	**%90.3**
Ortalama	**%89.389**	**%87.876**	**%86.442**	**%86.730**
Std. Sapma	**%3.906**	**%7.097**	**%7.070**	**%7.845**
En İyi	**%93.3**	**%92.3**	**%92.3**	**%92.0**
En Kötü	**%71.726**	**%67.230**	**%67.479**	**%67.476**
Medyan	**%90.833**	**%89.0**	**%89.333**	**%90.833**

DE/best/1 Diferansiyel Gelişim Algoritmasında Kullanılan Parametreler :

- Popülasyon Boyutu : 100
- Jenerasyon Sayısı : 2000
- F : 0.1;
- Çaprazlama/Mutasyon oranı : %90

	Parametre seti 1	Parametre seti 2	Parametre seti 3	parametre seti 4
Run 1	**%70.6**	**%72.3**	**%72.3**	**%72.3**
Run 2	**%70.3**	**%67.3**	**%72.0**	**%72.0**
Run 3	**%70.0**	**%67.0**	**%68.6**	**%71.3**
Run 4	**%67.3**	**%70.0**	**%68.3**	**%70.6**
Run 5	**%68.3**	**%72.3**	**%72.0**	**%67.3**
Run 6	**%70.6**	**%68.3**	**%69.3**	**%68.6**
Run 7	**%70.6**	**%69.3**	**%71.3**	**%72.3**
Run 8	**%71.3**	**%70.6**	**%69.3**	**%70.6**
Run 9	**%71.6**	**%69.6**	**%68.3**	**%69.3**
Run 10	**%69.3**	**%71.3**	**%69.0**	**%71.3**
Run 11	**%68.6**	**%69.3**	**%67.6**	**%68.3**
Run 12	**%68.3**	**%70.0**	**%70.0**	**%68.0**
Run 13	**%70.0**	**%70.0**	**%69.0**	**%70.0**
Run 14	**%69.3**	**%69.3**	**%72.6**	**%70.0**
Run 15	**%71.3**	**%69.3**	**%72.0**	**%70.0**
Run 16	**%71.6**	**%70.0**	**%72.6**	**%67.0**
Run 17	**%68.3**	**%72.0**	**%68.3**	**%69.0**
Run 18	**%68.0**	**%72.6**	**%68.6**	**%68.6**
Run 19	**%68.6**	**%72.3**	**%69.0**	**%69.6**
Run 20	**%69.3**	**%71.3**	**%69.3**	**%68.6**
Run 21	**%70.3**	**%71.0**	**%70.3**	**%71.0**
Run 22	**%68.0**	**%72.3**	**%67.6**	**%69.6**
Run 23	**%69.6**	**%72.0**	**%70.6**	**%70.6**
Run 24	**%69.0**	**%70.3**	**%70.0**	**%70.6**
Run 25	**%70.0**	**%71.0**	**%71.6**	**%70.0**
Ortalama	**%69.639**	**%70.453**	**%70.013**	**%69.893**
Std. Sapma	**%1.214**	**%1.531**	**%1.598**	**%1.425**
En İyi	**%71.6**	**%72.6**	**%72.6**	**%72.3**
En Kötü	**%67.3**	**%67.0**	**%67.6**	**%67.0**
Medyan	**%69.16**	**%70.0**	**%69.5**	**%69.0**

DE/best/2 Diferansiyel Gelişim Algoritmasında Kullanılan Parametreler :

- Popülasyon Boyutu : 100
- Jenerasyon Sayısı : 2000
- F : 0.1;
- Çaprazlama/Mutasyon oranı : %90

	Parametre seti 1	Parametre seti 2	Parametre seti 3	parametre seti 4
Run 1	**%72.0**	**%69.6**	**%69.3**	**%72.3**
Run 2	**%71.6**	**%72.0**	**%72.0**	**%68.6**
Run 3	**%70.0**	**%70.3**	**%69.3**	**%70.3**
Run 4	**%71.6**	**%69.6**	**%70.3**	**%70.3**
Run 5	**%70.6**	**%70.6**	**%68.3**	**%72.3**
Run 6	**%73.0**	**%70.6**	**%67.6**	**%73.6**
Run 7	**%68.6**	**%73.3**	**%69.3**	**%69.0**
Run 8	**%68.6**	**%74.3**	**%72.6**	**%70.3**
Run 9	**%72.6**	**%71.3**	**%69.6**	**%74.6**
Run 10	**%70.6**	**%70.6**	**%68.6**	**%68.0**
Run 11	**%68.6**	**%70.6**	**%69.0**	**%70.3**
Run 12	**%74.6**	**%73.3**	**%71.0**	**%70.3**
Run 13	**%73.3**	**%69.6**	**%72.0**	**%70.3**
Run 14	**%71.6**	**%70.6**	**%68.3**	**%69.3**
Run 15	**%69.3**	**%70.6**	**%74.3**	**%69.6**
Run 16	**%72.3**	**%70.3**	**%69.6**	**%72.0**
Run 17	**%73.6**	**%70.3**	**%73.6**	**%70.3**
Run 18	**%72.3**	**%71.6**	**%70.0**	**%68.6**
Run 19	**%68.6**	**%71.6**	**%73.0**	**%71.3**
Run 20	**%71.6**	**%70.0**	**%70.3**	**%72.3**
Run 21	**%73.6**	**%70.0**	**%70.3**	**%71.6**
Run 22	**%68.3**	**%72.3**	**%72.3**	**%70.0**
Run 23	**%70.6**	**%72.3**	**%70.6**	**%70.3**
Run 24	**%70.3**	**%71.3**	**%72.0**	**%70.3**
Run 25	**%71.0**	**%74.0**	**%70.6**	**%69.6**
Ortalama	**%71.2**	**%71.3**	**%70.586**	**%70.653**
Std. Sapma	**%1.773**	**%1.375**	**%1.728**	**%1.544**
En İyi	**%74.6**	**%74.3**	**%74.3**	**%74.6**
En Kötü	**%68.3**	**%69.6**	**%67.6**	**%68.0**
Medyan	**%74.0**	**%71.5**	**%71.5**	**%70.3**

DE/current_to_best/1 Diferansiyel Gelişim Algoritmasında Kullanılan Parametreler :

- Popülasyon Boyutu : 100
- Jenerasyon Sayısı : 2000
- F : 0.1;
- Çaprazlama/Mutasyon oranı : %90

	Parametre seti 1	Parametre seti 2	Parametre seti 3	parametre seti 4
Run 1	**%70.3**	**%71.3**	**%70.6**	**%67.6**
Run 2	**%69.6**	**%70.3**	**%70.0**	**%70.6**
Run 3	**%65.3**	**%68.0**	**%70.3**	**%68.0**
Run 4	**%69.3**	**%71.0**	**%68.6**	**%70.6**
Run 5	**%70.6**	**%71.3**	**%71.0**	**%68.3**
Run 6	**%69.3**	**%69.3**	**%73.0**	**%72.0**
Run 7	**%68.0**	**%71.6**	**%70.0**	**%73.0**
Run 8	**%70.6**	**%73.6**	**%68.0**	**%70.3**
Run 9	**%69.6**	**%71.0**	**%70.0**	**%72.6**
Run 10	**%68.6**	**%69.0**	**%70.6**	**%71.0**
Run 11	**%68.3**	**%68.3**	**%70.0**	**%69.6**
Run 12	**%69.6**	**%70.3**	**%69.3**	**%69.3**
Run 13	**%71.0**	**%68.6**	**%71.0**	**%70.0**
Run 14	**%68.6**	**%68.6**	**%68.0**	**%68.0**
Run 15	**%71.0**	**%68.6**	**%70.0**	**%68.0**
Run 16	**%70.6**	**%71.6**	**%70.0**	**%68.6**
Run 17	**%69.3**	**%70.0**	**%71.3**	**%68.3**
Run 18	**%71.3**	**%71.6**	**%70.0**	**%69.6**
Run 19	**%69.6**	**%70.3**	**%72.0**	**%70.3**
Run 20	**%71.0**	**%69.6**	**%71.6**	**%70.0**
Run 21	**%69.3**	**%67.6**	**%69.6**	**%71.3**
Run 22	**%72.3**	**%70.0**	**%71.6**	**%68.0**
Run 23	**%70.6**	**%70.3**	**%71.3**	**%69.3**
Run 24	**%70.3**	**%69.3**	**%70.6**	**%70.6**
Run 25	**%68.6**	**%71.0**	**%69.3**	**%67.6**
Ortalama	**%69.746**	**%70.12**	**%70.3**	**%69.73**
Std. Sapma	**%1.367**	**%1.385**	**%1.158**	**%1.523**
En İyi	**%72.3**	**%73.6**	**%73.0**	**%73.0**
En Kötü	**%65.3**	**%67.6**	**%68.0**	**%67.6**
Medyan	**%70.3**	**%69.5**	**%70.166**	**%69.6**

- **Genetik Algoritma Tablosunun Yorumu:**

Parametre optimizasyonu sırasında yüksek çaprazlama oranları sağlamanın modelin performansını arttırdığını saptadık. Bu sayede yaklaşık **%82.5** civarında müşteriyi kapsamına alabilen bir baz istasyonu yerleşim optimizasyonu gerçekleştirebildik. Belirlediğimiz mutasyon oranı da bu optimizasyona ulaşılması için geçen jenerasyon sayısının minimize edilmesinde oldukça etkili oldu. Ancak deneylerimiz sırasında çok yüksek mutasyon oranlarının popülasyonun genel stabilitesi açısından negatif olduğunu gözlemlediğimiz için oranı kabul edilebilir bir seviyede tuttuk. Ortalama **%55** civarı bir kapsama kapasitesinden **+%80** kapsama oranına yönelik bir optimizasyon gerçekleştirmeyi başardık.

- **Diferansiyel Gelişim Tablolarının Yorumu:**

Tüm mutasyon algoritmaları eşdeğer performans göstermedi. DE/rand/1 ve DE/rand/2 mutasyon fonksiyonları en iyi performansı göstermek ile birlikte yaklaşık **%88-89** civarında müşteriyi kapsayarak sistemi optimize etmeyi başardı. DE/best/2 performans açısından DE/rand/ fonksiyonlarına göre daha verimsiz sonuçlar elde ederek yaklaşık **%70-71** kapsama oranına erişebildi. En kötü performansı ise DE/best/1 ve DE/current_to_best/1 mutasyon fonksiyonları sergiledi. Yaklaşık **%69-70** arası ortalama kapsama performansına erişebildiler.

Ulaşılabilen maksimum kapsama başarılarından en iyisini ise **%93.3** kapsama oranı ile DE/rand/2 sağladı. Bunun ardından **%92.3** kapsama oranı ile DE/rand/1 takip ediyor. DE/best/2 ve DE/current_to_best/1 mutasyon fonksiyonları **%74.6** ve **%73.6** maksimum kapsama oranlarıyla performans açısından oldukça geride kaldı. Ortalama değerlerde en kötü başarıyı sergileyen DE/best/1 ise **%72.6** kapsama oranına erişebildi.

Standart sapmalar incelendiğinde çözümler içerisindeki istikrarın kabul edilebilir bir düzeyde olduğunu söyleyebiliriz. Sistemin DE/best/1 , DE/best/2 ve DE/current_to_best/1 mutasyon fonksiyonları ile iyi çalışmamasının sebepleri arasında hiper parametrelerin ortak olmasının rol oynadığını düşünüyoruz. Daha yüksek mutasyon ve harmanlanma oranlarıyla bağımsız test sınıflarında daha iyi sonuçlar elde etmeyi başardık.

05170000006 – Ege Doğan Dursun
05130000242 – Cem Çorbacıoğlu