LAMPIRAN

Program aplikasi dengan random input host

Class Host :

```
public class Host {
   private int xPos;
   private int yPos;
   private int id;

   public Host(int x, int y, int i) {
      xPos = x;
      yPos = y;
      id = i;
   }

   public int getX(){
      return xPos;
   }

   public int getY(){
      return yPos;
   }

   public int getId(){
      return id;
   }

   public int proximity(int x, int y){
      int xdiff = xPos - x;
      int ydiff = yPos - y;
      return (int)Math.sqrt( xdiff*xdiff + ydiff*ydiff );
   }
```

```
    public int proximity(Host other){
        return proximity(other.getX(), other.getY());
    }
}
```

Class Chromosome :

```
public class Chromosome{
        // kumpulan dari host/node/titik, yang merupakan kumpulan gen dari
kromosom
        protected int[] hostList;
        // nilai atau harga dari hostList
        protected double cost;
        // persentase mutasi
        protected double mutation;
        // variable menampung banyaknya genetic algoritm digunakan
        protected int cutLength;
        /**
        * user-defined constructor
        *
        * @param hosts merepresentasikan gen-gen pada kromosom
        */
        public Chromosome(Host[] hosts) {
            boolean taken[] = new boolean[hosts.length];
            hostList = new int[hosts.length];
            cost = 0.0;
            for ( int i=0;i<hostList.length;i++ ) taken[i] = false;
            for ( int i=0;i<hostList.length-1;i++ ) {
                int icandidate;
                do {
                    icandidate = (int) ( 0.999999* Math.random() *
                            (double) hostList.length );
```

```
        } while ( taken[icandidate] );
        hostList[i] = icandidate;
        taken[icandidate] = true;
        if ( i == hostList.length-2 ) {
            icandidate = 0;
            while ( taken[icandidate] ) icandidate++;
            hostList[i+1] = icandidate;
        }
    }
    calculateCost(hosts);
    cutLength = 1;
}

public void calculateCost(Host[] hosts){
    cost=0;
    for ( int i=0;i<hostList.length-1;i++ ) {
        double dist = hosts[hostList[i]].proximity(hosts[hostList[i+1]]);
        cost += dist;
    }
}

public double getCost(){
    return cost;
}

public int getHost(int n){
    return hostList[n];
}

public void setHosts(int[] list){
    for(int i = 0; i < hostList.length; i++){
        hostList[i] = list[i];
```

```
    }
  }

  public void setHost(int index, int number){
    hostList[index] = number;
  }

  public void setGA(int i){
    cutLength = i;
  }

  public void setMutation(double prob){
    mutation = prob;
  }


 public int mate(Chromosome father, Chromosome offspring1, Chromosome
offspring2){
  int cutpoint1 = (int) (0.999999*Math.random()*(double)(hostList.length-
cutLength));
  int cutpoint2 = cutpoint1 + cutLength;

  boolean taken1 [] = new boolean[hostList.length];
  boolean taken2 [] = new boolean[hostList.length];
  int off1 [] = new int[hostList.length];
  int off2 [] = new int[hostList.length];

  for ( int i=0;i<hostList.length;i++ ) {
    taken1[i] = false;
    taken2[i] = false;
  }
```

```
for ( int i=0;i<hostList.length;i++ ) {
   if ( i<cutpoint1 || i>= cutpoint2 ) {
      off1[i] = -1;
      off2[i] = -1;
   } else {
      int imother = hostList[i];
      int ifather = father.getHost(i);
      off1[i] = ifather;
      off2[i] = imother;
      taken1[ifather] = true;
      taken2[imother] = true;
   }
}

for ( int i=0;i<cutpoint1;i++ ) {
   if ( off1[i] == -1 ) {
    for ( int j=0;j<hostList.length;j++ ) {
      int imother = hostList[j];
      if ( !taken1[imother] ) {
         off1[i] = imother;
         taken1[imother] = true;
         break;
      }
    }
   }
   if ( off2[i] == -1 ) {
      for ( int j=0;j<hostList.length;j++ ) {
         int ifather = father.getHost(j);
         if ( !taken2[ifather] ) {
            off2[i] = ifather;
            taken2[ifather] = true;
            break;
```

```
            }
          }
        }
      }
    for ( int i=hostList.length-1;i>=cutpoint2;i-- ) {
      if ( off1[i] == -1 ) {
          for ( int j=hostList.length-1;j>=0;j-- ) {
            int imother = hostList[j];
            if ( !taken1[imother] ) {
              off1[i] = imother;
              taken1[imother] = true;
              break;
            }
          }
        }
        if ( off2[i] == -1 ) {
          for ( int j=hostList.length-1;j>=0;j-- ) {
            int ifather = father.getHost(j);
            if ( !taken2[ifather] ) {
              off2[i] = ifather;
              taken2[ifather] = true;
              break;
            }
          }
        }
      }

      offspring1.setHosts(off1);
      offspring2.setHosts(off2);

      int mutate = 0;
      if ( Math.random() < mutation ) {
```

```
        int iswap1 = (int) (0.999999*Math.random()*(double)(hostList.length));
        int iswap2 = (int) (0.999999*Math.random()*(double)hostList.length);
        int i = off1[iswap1];
        off1[iswap1] = off1[iswap2];
        off1[iswap2] = i;
        mutate++;
    }
    if ( Math.random() < mutation ) {
        int iswap1 = (int) (0.999999*Math.random()*(double)(hostList.length));
        int iswap2 = (int) (0.999999*Math.random()*(double)hostList.length);
        int i = off2[iswap1];
        off2[iswap1] = off2[iswap2];
        off2[iswap2] = i;
        mutate++;
    }
    return mutate;
}

    public static void sortChromosome(Chromosome chromosomes[], int num){
        Chromosome ctemp;
        boolean swapped = true;
        while ( swapped ) {
        swapped = false;
        for ( int i=0;i<num-1;i++ ) {
            if ( chromosomes[i].getCost() > chromosomes[i+1].getCost() ) {
                ctemp = chromosomes[i];
                chromosomes[i] = chromosomes[i+1];
                chromosomes[i+1] = ctemp;
                swapped = true;
            }
        }
        }
```

```
        }
}
```

Class SimulationTSP :

```
import java.awt.event.*;
import java.awt.*;
import java.io.IOException;
import java.text.*;
import javax.swing.*;
/**
 * kelas ini adalah kelas aplikasi simulasi pencarian jalur optimal dalam sebuah
 * jaringan dengan menggunakan Genetic Algoritms
 */
public class SimulationTSPApp extends Frame implements Runnable,
ActionListener{
    private Canvas canvas;
    //private JFrame frame;
    private Button start, exit, generate;
    private TextField host, population, percentMutation;
    private Label labelHost, labelPopulation, labelPercentMutation;
    private Panel panel;

    protected int hostCount;
    protected int populationSize;
    protected double mutationPercent;
    protected int matingPopulationSize;
    protected int favorPopulationSize;
    protected int cutLength;
    protected int generation;
    protected Thread work = null;
    protected boolean started = false;
```

```
protected Host[] hosts;

protected Chromosome[] chromosomes;
private String status = "";

public SimulationTSPApp() {
    //frame = new JFrame("Simulasi Jalur Optimal using Genetic Algoritm");
    setLayout(new BorderLayout());
    setTitle("Simulasi Jalur Optimal dengan ALgoritma Genetic");
    setSize(700,500);
    canvas = new Canvas();
    //canvas.setSize(700,400);
    canvas.setBackground(Color.WHITE);
    panel = new Panel();
    generate = new Button("Generate Host");
    generate.addActionListener(this);
    panel.add(generate);
    start = new Button("Start");
    start.addActionListener(this);
    panel.add(start);
    labelHost = new Label("**Hosts");
    panel.add(labelHost);
    host = new TextField(5);
    panel.add(host);
    labelPopulation = new Label("*Population size");
    panel.add(labelPopulation);
    population = new TextField(5);
    panel.add(population);
    labelPercentMutation = new Label(" % mutate");
    panel.add(labelPercentMutation);
    percentMutation = new TextField(5);
    panel.add(percentMutation);
```

```
    exit = new Button("Exit");
    exit.addActionListener(this);
    panel.add(exit);

    //finishing build the UI

    this.add(panel, BorderLayout.SOUTH);
    this.add(canvas, BorderLayout.CENTER);
    //this.pack();
    //this.setDefaultCloseOperation(this.EXIT_ON_CLOSE);
    this.setVisible(true);

    //set default value to start the simulation
    host.setText("10");
    population.setText("100");
    percentMutation.setText("0.10");

    started = false;
    update();
}

public void startThread(){
    try{
        hostCount = Integer.parseInt(host.getText());
    }catch(Exception ioe){
        JOptionPane.showMessageDialog(null,"Masukan nilai integer " +
                "untuk menentukan jumlah host");
    }
    try{
        populationSize = Integer.parseInt(population.getText());
    }catch(Exception ioe){
        JOptionPane.showMessageDialog(null,"Masukan nilai integer " +
```

```
            "untuk menentukan jumlah populasi");
}
try{
   mutationPercent = new Double(percentMutation.getText()).doubleValue();
}catch(Exception ioe){
   JOptionPane.showMessageDialog(null,"Masukan nilai double " +
            "untuk menentukan persentase mutasi");
}

FontMetrics fm = canvas.getGraphics().getFontMetrics();
int bottom = panel.getBounds().y - fm.getHeight()-2;

matingPopulationSize = populationSize/2;
favorPopulationSize = matingPopulationSize/2;
cutLength = hostCount/5;

//random list of hosts
hosts = new Host[hostCount];
    for ( int i=0;i<hostCount;i++ ) {
      hosts[i] = new Host(
        (int)(Math.random()*(canvas.getBounds().width-10)),
        (int)(Math.random()*(bottom-10)), i+1);
    }


//inisialisasi chromosome
chromosomes = new Chromosome[populationSize];
    for ( int i=0;i<populationSize;i++ ) {
      chromosomes[i] = new Chromosome(hosts);
      chromosomes[i].setGA(cutLength);
      chromosomes[i].setMutation(mutationPercent);
    }
```

```
            Chromosome.sortChromosome(chromosomes,populationSize);


        //start up background thread
        started = true;

        generation = 0;

        update();
        /*
        if ( work != null )
           work = null;
        work = new Thread(this);
        work.setPriority(Thread.MIN_PRIORITY);
        work.start();
        */
    }

    public void update(){
        Image img = createImage(canvas.getBounds().width,
canvas.getBounds().height);
        Graphics g = img.getGraphics();
        FontMetrics fm = g.getFontMetrics();

        int width = canvas.getBounds().width;
        int bottom = panel.getBounds().y - fm.getHeight()-2;

        g.setColor(Color.GRAY);
        g.fillRect(0, 0, width, bottom);

        if( started && (hosts != null) )
        {
```

```
            g.setColor(Color.BLACK);
            for ( int i=0;i<hostCount;i++ ) {
              int xpos = hosts[i].getX();
              int ypos = hosts[i].getY();
             int id = hosts[i].getId();
              g.fillOval(xpos-5,ypos-5,10,10);
             g.drawString(" "+id, hosts[i].getX(), hosts[i].getY());
            }

            g.setColor(Color.BLUE);
            for ( int i=0;i<hostCount;i++ ) {
              int icity = chromosomes[0].getHost(i);
              if ( i!=0 ) {
                int last = chromosomes[0].getHost(i-1);
                g.drawLine(
                        hosts[icity].getX(),
                        hosts[icity].getY(),
                        hosts[last].getX(),
                        hosts[last].getY());
               try{
                  WriteToText.addText(""+hosts[icity].getId()+"
("+hosts[icity].getX()+","+hosts[icity].getY()+")"+","+" "+hosts[last].getId()+"
("+hosts[last].getX()+","+hosts[last].getY()+")"); }catch(IOException ioe){}
             }
            }

        }


        //g.drawString(status, 0, bottom);

        canvas.getGraphics().drawImage(img, 0, 0, canvas);
```

```
    }

    public void setStatus(String status)
    {
       this.status = status;
    }

    public void run(){
       double thisCost = 500.0;
       double oldCost = 0.0;
       double dcost = 500.0;
       int countSame = 0;

       update();

       while(countSame<100) {

       generation++;

       int ioffset = matingPopulationSize;
       int mutated = 0;


      // Mate the chromosomes in the favoured population
      // with all in the mating population
       for ( int i=0;i<favorPopulationSize;i++ ) {
          Chromosome cmother = chromosomes[i];
          // Select partner from the mating population
          int father = (int) (
0.999999*Math.random()*(double)matingPopulationSize);
          Chromosome cfather = chromosomes[father];
```

```
            mutated +=
cmother.mate(cfather,chromosomes[ioffset],chromosomes[ioffset+1]);
            ioffset += 2;
        }

        // The new generation is in the matingPopulation area
        // move them to the correct area for sort.
        for ( int i=0;i<matingPopulationSize;i++ ) {
            chromosomes[i] = chromosomes[i+matingPopulationSize];
            chromosomes[i].calculateCost(hosts);
        }


       // Now sort the new mating population
        Chromosome.sortChromosome(chromosomes,matingPopulationSize);

        double cost = chromosomes[0].getCost();
        dcost = Math.abs(cost-thisCost);
        thisCost = cost;
        double mutationRate = 100.0 * (double) mutated / (double)
matingPopulationSize;

        NumberFormat nf = NumberFormat.getInstance();
        nf.setMinimumFractionDigits(2);
        nf.setMinimumFractionDigits(2);
        try{
            WriteToText.addText("Generation "+generation+" Cost "+(int)thisCost+"
Mutated "+nf.format(mutationRate)+"%");
        }catch(IOException ioe){}

        if ( (int)thisCost == (int)oldCost ) {
```

```
            countSame++;
        } else {
            countSame = 0;
            oldCost = thisCost;
        }
        update();



        }
        try{
            WriteToText.addText("Solution found after "+generation+"
generations.\n");
        }catch(IOException ioe){}
    }

    public void paint(Graphics g){
        update();
    }

    public void actionPerformed(ActionEvent e){
        Object source = e.getSource();
        if(source == exit){
            System.exit(0);
        }
        if(source == start){
            FontMetrics fm = canvas.getGraphics().getFontMetrics();
            int bottom = panel.getBounds().y - fm.getHeight()-2;

            matingPopulationSize = populationSize/2;
            favorPopulationSize = matingPopulationSize/2;
            cutLength = hostCount/5;
```

```
            //inisialisasi chromosome
            chromosomes = new Chromosome[populationSize];
            for ( int i=0;i<populationSize;i++ ) {
            chromosomes[i] = new Chromosome(hosts);
            chromosomes[i].setGA(cutLength);
            chromosomes[i].setMutation(mutationPercent);
            }
            Chromosome.sortChromosome(chromosomes,populationSize);


            //start up background thread
            started = true;

            generation = 0;
            if ( work != null )
               work = null;
            work = new Thread(this);
            work.setPriority(Thread.MIN_PRIORITY);
            work.start();
         }
         if(source == generate){
            startThread();
         }
      }
      public static void main(String[] args){
         SimulationTSPApp application = new SimulationTSPApp();
      }
}
```

Class Write to Text :

```
import java.io.*;
public class WriteToText {
```

```
    public WriteToText() {
    }
    public static void addText(String str) throws IOException{

        PrintWriter out = new PrintWriter(
                new FileWriter("Test.txt",true));
        out.println(str);
        out.close();
    }
}
```

Program aplikasi dengan manual input host

```
package GA;

import java.awt.event.*;
import java.awt.*;
import java.io.IOException;
import java.io.*;
import java.text.*;
import javax.swing.*;
/**
 *
 * kelas ini adalah kelas aplikasi simulasi pencarian jalur optimal dalam sebuah
 * jaringan dengan menggunakan Genetic Algoritms
 */
public class SimulationTSPApp extends Frame implements Runnable,
ActionListener, MouseListener, MouseMotionListener{
    /**
     * komponen-komponen untuk membuat GUI applikasi
     */
    private Grids canvas;
    //private JFrame frame;
    private Button start, exit, generate;
    private TextField host, population, percentMutation;
    private Label labelHost, labelPopulation, labelPercentMutation;
    private Panel panel;

    protected int originX, originY;

    /**
     * variable untuk menampung berapa banyak host atau nodes yang digunakan
     */
```

```
protected int hostCount;

    // host tracker, tracking nodes created with mouse click event
protected int hostTracker;
/**
 * menampung banyaknya populasi
 */
protected int populationSize;
/**
 * persentase yang digunakan untuk mutasi
 */
protected double mutationPercent;
/**
 * menampung bagian populasi yang memenuhi syarat untuk crossover
 */
protected int matingPopulationSize;
/**
 * menampung hasil populasi dari crossover
 */
protected int favorPopulationSize;
/**
 * menampung berapa banyak generate genetic sewaktu proses crossover
 */
protected int cutLength;
/**
 * menampung jumlah generasi
 */
protected int generation;
/**
 * menciptakan thread untuk background process
 */
protected Thread work = null;
```

```
/**
 * variable untuk menentukan thread tersebut dimulai atau belum
 */
protected boolean started = false;
/**
 * kumpulan dari hosts
 */
protected Host[] hosts;
/**
 * kumpulan chromosome
 */
protected Chromosome[] chromosomes;
/**
 * menampuang informasi status dari genetic algoritm
 */
private String status = "";

public SimulationTSPApp() {
    /**
     * instansiasi graphic user interface yang akan dibangun
     */
    //frame = new JFrame("Simulasi Jalur Optimal dengan Genetic Algoritm");
    setLayout(new BorderLayout());
    setTitle("Simulasi Jalur Optimal dengan Genetic Algoritm");
    setSize(700,500);
    canvas = new Grids(700,450,40,55);
    //canvas.setSize(700,400);
    canvas.setBackground(Color.WHITE);


    canvas.addMouseListener(this);
    canvas.addMouseMotionListener(this);
```

```
panel = new Panel();
generate = new Button("Generate Host");
generate.addMouseListener(this);
panel.add(generate);
start = new Button("Start");
start.addActionListener(this);
panel.add(start);
labelHost = new Label("**Hosts");
panel.add(labelHost);
host = new TextField(2);
panel.add(host);
labelPopulation = new Label("*Population size");
panel.add(labelPopulation);
population = new TextField(5);
panel.add(population);
labelPercentMutation = new Label(" % mutate");
panel.add(labelPercentMutation);
percentMutation = new TextField(5);
panel.add(percentMutation);
exit = new Button("Exit");
exit.addActionListener(this);
panel.add(exit);

//finishing build the UI

this.add(panel, BorderLayout.SOUTH);
this.add(canvas, BorderLayout.CENTER);

//this.pack();
//this.setDefaultCloseOperation(this.EXIT_ON_CLOSE);
this.setVisible(true);
```

```
    //set default value to start the simulation
    host.setText("10");
    population.setText("100");
    percentMutation.setText("0.10");

    started = false;
    update();
}

public void startThread(){
    try{
        hostCount = Integer.parseInt(host.getText());
    }catch(Exception ioe){
        JOptionPane.showMessageDialog(null,"Masukan nilai integer " +
                "untuk menentukan jumlah host");
    }
    try{
        populationSize = Integer.parseInt(population.getText());
    }catch(Exception ioe){
        JOptionPane.showMessageDialog(null,"Masukan nilai integer " +
                "untuk menentukan jumlah populasi");
    }
    try{
        mutationPercent = new Double(percentMutation.getText()).doubleValue();
    }catch(Exception ioe){
        JOptionPane.showMessageDialog(null,"Masukan nilai double " +
                "untuk menentukan persentase mutasi");
    }

    FontMetrics fm = canvas.getGraphics().getFontMetrics();
    int bottom = panel.getBounds().y - fm.getHeight()-2;
```

```
matingPopulationSize = populationSize/2;
favorPopulationSize = matingPopulationSize/2;
cutLength = hostCount/5;

if(hosts==null){
hosts = new Host[hostCount];
    for ( int i=0;i<hostCount;i++ ) {
                  hosts[i] = new Host(
                      (int)(Math.random()*(canvas.getBounds().width-10)),
                      (int)(Math.random()*(bottom-10)), i+1);
                }

    }

//inisialisasi chromosome

chromosomes = new Chromosome[populationSize];
    for ( int i=0;i<populationSize;i++ ) {
      chromosomes[i] = new Chromosome(hosts);
      chromosomes[i].setGA(cutLength);
      chromosomes[i].setMutation(mutationPercent);
    }
    Chromosome.sortChromosome(chromosomes,populationSize);


//start up background thread
started = true;

generation = 0;

update();
```

```
        if ( work != null )
            work = null;
        work = new Thread(this);
        work.setPriority(Thread.MIN_PRIORITY);
        work.start();

    }
    /**
     * method untuk update tampilan
     */
    public void mouseClicked(MouseEvent e){

        if(e.getSource().equals(canvas)){

            if(hosts==null){
                        try{
                            hostCount = Integer.parseInt(host.getText());
                            hosts = new Host[hostCount];
                            hostTracker = 0;
                          }
                        catch(Exception ioe){
                              JOptionPane.showMessageDialog(null,"Masukan
nilai integer " +
                                    "untuk menentukan jumlah host");
                    }
                    }
            if(hostTracker<hostCount){
            originX = e.getX() - 5;
            originY = e.getY() - 5;
            Graphics g = canvas.getGraphics();
            g.fillOval(originX, originY, 5, 5);
```

```
            hosts[hostTracker] = new Host(originX, originY, hostTracker+1);
                        hostTracker++;
            }

        }

    }

    public void update(){
        Image img = createImage(canvas.getBounds().width,
canvas.getBounds().height);
        Graphics g = img.getGraphics();
        FontMetrics fm = g.getFontMetrics();

        int width = canvas.getBounds().width;
        //int height = canvas.getBounds().height;
        int bottom = panel.getBounds().y - fm.getHeight()-2;

        //int rows, columns;

        g.setColor(Color.GRAY);
        g.fillRect(0, 0, width, bottom);

        if( started && (hosts != null) )
        {
            g.setColor(Color.BLACK);
            for ( int i=0;i<hostCount;i++ ) {
              int xpos = hosts[i].getX();
              int ypos = hosts[i].getY();
             int id = hosts[i].getId();
              g.fillOval(xpos-5,ypos-5,10,10);
```

```
            g.drawString(" "+id, hosts[i].getX(), hosts[i].getY());
           }

           g.setColor(Color.BLUE);
           for ( int i=0;i<hostCount;i++ ) {
            int icity = chromosomes[0].getHost(i);
            if ( i!=0 ) {
              int last = chromosomes[0].getHost(i-1);
              g.drawLine(
                      hosts[icity].getX(),
                      hosts[icity].getY(),
                      hosts[last].getX(),
                      hosts[last].getY());
             try{
                 WriteToText.addText(""+hosts[icity].getId()+"
("+hosts[icity].getX()+","+hosts[icity].getY()+")"+","+" "+hosts[last].getId()+"
("+hosts[last].getX()+","+hosts[last].getY()+")"); }catch(IOException ioe){}
            }
           }

       }

       //g.drawString(status, 0, bottom);

       canvas.getGraphics().drawImage(img, 0, 0, canvas);
       canvas.paint(g);

   }
   /**
    * method untuk set status dari GA
    */
```

```
    public void setStatus(String status)
    {
       this.status = status;
    }
    /**
     * method run ini yang akan dikerjakan oleh thread
     */

    public void run(){
       double thisCost = 500.0;
       double oldCost = 0.0;
       double dcost = 500.0;
       int countSame = 0;

       //update();

       while(countSame<100) {

       generation++;

       int ioffset = matingPopulationSize;
       int mutated = 0;

      // Mate the chromosomes in the favoured population
      // with all in the mating population
       for ( int i=0;i<favorPopulationSize;i++ ) {
          Chromosome cmother = chromosomes[i];
          // Select partner from the mating population
          int father = (int) (
0.999999*Math.random()*(double)matingPopulationSize);
          Chromosome cfather = chromosomes[father];
```

```
            mutated +=
cmother.mate(cfather,chromosomes[ioffset],chromosomes[ioffset+1]);
            ioffset += 2;
        }

        // The new generation is in the matingPopulation area
        // move them to the correct area for sort.
        for ( int i=0;i<matingPopulationSize;i++ ) {
            chromosomes[i] = chromosomes[i+matingPopulationSize];
            chromosomes[i].calculateCost(hosts);
        }

       // Now sort the new mating population
        Chromosome.sortChromosome(chromosomes,matingPopulationSize);

        double cost = chromosomes[0].getCost();
        dcost = Math.abs(cost-thisCost);
        thisCost = cost;
        double mutationRate = 100.0 * (double) mutated / (double)
matingPopulationSize;

        NumberFormat nf = NumberFormat.getInstance();
        nf.setMinimumFractionDigits(2);
        nf.setMinimumFractionDigits(2);
        try{
            WriteToText.addText("Generation "+generation+" Cost "+(int)thisCost+"
Mutated "+nf.format(mutationRate)+"%");
        }catch(IOException ioe){}

        if ( (int)thisCost == (int)oldCost ) {
            countSame++;
        } else {
```

```
            countSame = 0;
            oldCost = thisCost;
        }
        update();



        }
        try{
            WriteToText.addText("Solution found after "+generation+"
generations.\n");
        }catch(IOException ioe){}
    }
    /**
     * meng-handle aksi dari tombol yang ditekan
     */
    public void actionPerformed(ActionEvent e){

        Object source = e.getSource();

        if(source == exit){
            System.exit(0);
        }
        if(source == start){
            startThread();
        }
        /*
        if(source == generate){
            startThread();
        }*/
    }
    public static void main(String[] args){
        SimulationTSPApp application = new SimulationTSPApp();
```

```
    }

    public void mousePressed(MouseEvent e) {    }

    public void mouseReleased(MouseEvent e) {    }

    public void mouseEntered(MouseEvent e) {    }

    public void mouseExited(MouseEvent e) {    }

    public void mouseDragged(MouseEvent e) {    }

    public void mouseMoved(MouseEvent e) {    }
}

class Chromosome{
      // kumpulan dari host/node/titik, yang merupakan kumpulan gen dari
kromosom
      protected int[] hostList;
      // nilai atau harga dari hostList
      protected double cost;
      // persentase mutasi
      protected double mutation;
      // variable menampung banyaknya genetic algoritm digunakan
      protected int cutLength;
      /**
      * user-defined constructor
      *
      * @param hosts merepresentasikan gen-gen pada kromosom
      */
      public Chromosome(Host[] hosts) {
         boolean taken[] = new boolean[hosts.length];
```

```
            hostList = new int[hosts.length];
            cost = 0.0;
            for ( int i=0;i<hostList.length;i++ ){
                                    taken[i] = false;
                        }
            for ( int i=0;i<hostList.length-1;i++ ) {
                                    int icandidate;
                                    do {
                                            icandidate = (int) ( 0.999999*
Math.random() *
                                                                    (double)
hostList.length );
                                    } while ( taken[icandidate] );
                                    hostList[i] = icandidate;
                                    taken[icandidate] = true;
                                    if ( i == hostList.length-2 ) {
                                            icandidate = 0;
                                            while ( taken[icandidate] ) icandidate++;
                                            hostList[i+1] = icandidate;
                                    }

            }
            calculateCost(hosts);
            cutLength = 1;
        }
        /**
        *  method untuk menghitung nilai atau harga/cost dari kumpulan host
        *  @param hosts kumpulan dari host
        */
        public void calculateCost(Host[] hosts){
            cost=0;
            for ( int i=0;i<hostList.length-1;i++ ) {
```

```
                                    if(hosts[i]!=null){
                                    double dist =
hosts[hostList[i]].proximity(hosts[hostList[i+1]]);
                                    cost += dist;
                                }
        }
    }
    /**
    * mengembalikan nilai atau harga dari kromosom.
    * nilai atau harga ini yang nantinya menentukan apakah kromosom tersebut
    * dapat di-recombine (crossover) atau dieliminasi
    */
    public double getCost(){
        return cost;
    }
    /**
    * method untuk mendapatkan n-th host
    * contoh: nilai 3 akan menunjukan host ke-4 yang akan dilewati
    *         nilai 0 akan menunjukan host pertama yang akan dilewati
    * @param n host yang dituju
    */
    public int getHost(int n){
        return hostList[n];
    }
    /**
    * method untuk men-set urutan dari hosts yang akan dilewati
    * @param list kumpulan host
    */
    public void setHosts(int[] list){
        for(int i = 0; i < hostList.length; i++){
            hostList[i] = list[i];
        }
```

```
        }
        /**
        * method utk men-set index dari host
        * @param index index host
        * @param number id host yang akan menempati index
        */
        public void setHost(int index, int number){
            hostList[index] = number;
        }
        /**
        *  method utk setting ga.
        *  ga ini digunakan untuk mengetahui berapa kali genetic digunakan
        *  untuk tiap pasangan kromosom saat crossover
        */
        public void setGA(int i){
            cutLength = i;
        }
        /**
        * method untuk setting persentase mutasi
        * @param prob probabilitas yang akan digunakan untuk mutasi
        */
        public void setMutation(double prob){
            mutation = prob;
        }
    /**
     * method untuk melakukan crossover antar kromosom dan mutasi kromosom
     * anggap bahwa kelas Chromosome ini merepresentasikan kromosom 'ibu'
yang
     * akan berpasangan dengan kromosom 'bapak' yang dideklarasikan di param
method ini
     *
     * @param male kromosom 'bapak'
```

```
    * @param offspring1 offspring pertama
    * @param offspring2 offspring kedua
    * mengembalikan jumlah mutasi yang digunakan
    */
    public int mate(Chromosome father, Chromosome offspring1, Chromosome
offspring2){
      int cutpoint1 = (int) (0.999999*Math.random()*(double)(hostList.length-
cutLength));
      int cutpoint2 = cutpoint1 + cutLength;

      boolean taken1 [] = new boolean[hostList.length];
      boolean taken2 [] = new boolean[hostList.length];
      int off1 [] = new int[hostList.length];
      int off2 [] = new int[hostList.length];

      for ( int i=0;i<hostList.length;i++ ) {
         taken1[i] = false;
         taken2[i] = false;
      }

      for ( int i=0;i<hostList.length;i++ ) {
         if ( i<cutpoint1 || i>= cutpoint2 ) {
            off1[i] = -1;
            off2[i] = -1;
         } else {
            int imother = hostList[i];
            int ifather = father.getHost(i);
            off1[i] = ifather;
            off2[i] = imother;
            taken1[ifather] = true;
            taken2[imother] = true;
         }
```

```
  }

  for ( int i=0;i<cutpoint1;i++ ) {
     if ( off1[i] == -1 ) {
      for ( int j=0;j<hostList.length;j++ ) {
        int imother = hostList[j];
        if ( !taken1[imother] ) {
           off1[i] = imother;
           taken1[imother] = true;
           break;
        }
      }
     }
     if ( off2[i] == -1 ) {
        for ( int j=0;j<hostList.length;j++ ) {
           int ifather = father.getHost(j);
           if ( !taken2[ifather] ) {
              off2[i] = ifather;
              taken2[ifather] = true;
              break;
           }
        }
     }
  }
for ( int i=hostList.length-1;i>=cutpoint2;i-- ) {
  if ( off1[i] == -1 ) {
        for ( int j=hostList.length-1;j>=0;j-- ) {
           int imother = hostList[j];
           if ( !taken1[imother] ) {
              off1[i] = imother;
              taken1[imother] = true;
              break;
```

```
            }
        }
    }
    if ( off2[i] == -1 ) {
        for ( int j=hostList.length-1;j>=0;j-- ) {
            int ifather = father.getHost(j);
            if ( !taken2[ifather] ) {
                off2[i] = ifather;
                taken2[ifather] = true;
                break;
            }
        }
    }
}

offspring1.setHosts(off1);
offspring2.setHosts(off2);

int mutate = 0;
if ( Math.random() < mutation ) {
    int iswap1 = (int) (0.999999*Math.random()*(double)(hostList.length));
    int iswap2 = (int) (0.999999*Math.random()*(double)hostList.length);
    int i = off1[iswap1];
    off1[iswap1] = off1[iswap2];
    off1[iswap2] = i;
    mutate++;
}
if ( Math.random() < mutation ) {
    int iswap1 = (int) (0.999999*Math.random()*(double)(hostList.length));
    int iswap2 = (int) (0.999999*Math.random()*(double)hostList.length);
    int i = off2[iswap1];
    off2[iswap1] = off2[iswap2];
```

```
            off2[iswap2] = i;
            mutate++;
        }
        return mutate;
    }
        /**
         *  method mengurutkan kromosom berdasarkan cost
         *  @param chromosomes array dari kromosom yang akan diurutkan
         *  @param num berapa banyak list kromosom yang akan diurutkan
         */
        public static void sortChromosome(Chromosome chromosomes[], int num){
            Chromosome ctemp;
            boolean swapped = true;
            while ( swapped ) {
            swapped = false;
            for ( int i=0;i<num-1;i++ ) {
                if ( chromosomes[i].getCost() > chromosomes[i+1].getCost() ) {
                    ctemp = chromosomes[i];
                    chromosomes[i] = chromosomes[i+1];
                    chromosomes[i+1] = ctemp;
                    swapped = true;
                }
            }
            }

        }
}

class WriteToText {

    /** Creates a new instance of WriteToText */
    public WriteToText() {
```

```
    }
    public static void addText(String str) throws IOException{

        PrintWriter out = new PrintWriter(
                new FileWriter("Test.txt",true));
        out.println(str);
        out.close();
    }
}

class Host {
    private int xPos;
    private int yPos;
    private int id;
    /**
     * user-defined constructor
     * @param x assign the host's x position
     * @param y assign the host's y position
     */
    public Host(int x, int y, int i) {
        xPos = x;
        yPos = y;
        id = i;
    }

    public int getX(){
        return xPos;
    }

    public int getY(){
        return yPos;
    }
```

```
    public int getId(){
        return id;
    }
    /** return how far the host is from specific point
     *  this method uses the pythagorean to calculate the distance
     *  @param x the x coordinate
     *  @param y the y coordinate
     *  this method will return the distance
     */
    public int proximity(int x, int y){
        int xdiff = xPos - x;
        int ydiff = yPos - y;
        return (int)Math.sqrt( xdiff*xdiff + ydiff*ydiff );
    }
    /** return how close the host is to the others
     *  @param other the other host
     *  return a distance
     */
    public int proximity(Host other){
        return proximity(other.getX(), other.getY());
    }
}

class Grids extends Canvas{

        int width, height, rows, columns;

        Grids(int w, int h, int r, int c) {
            setSize(width = w, height = h);
            rows = r;
            columns = c;
```

```
        }
         public void paint(Graphics g) {
         int k;
         width = getSize().width;
         height = getSize().height;

         int htOfRow = height / (rows);
         for (k = 0; k < rows; k++)
           g.drawLine(0, k * htOfRow , width, k * htOfRow );

         int wdOfRow = width / (columns);
         for (k = 0; k < columns; k++)
           g.drawLine(k*wdOfRow , 0, k*wdOfRow , height);
  }
}
```

Filetext percobaan pada gambar 4.7 dan gambar 4.8

```
5 (183,246), 1 (379,78)
9 (365,153), 5 (183,246)
2 (443,69), 9 (365,153)
7 (324,309), 2 (443,69)
10 (624,331), 7 (324,309)
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
4 (119,297), 8 (669,384)
3 (33,251), 4 (119,297)
1 (379,78), 3 (33,251)
9 (365,153), 1 (379,78)
2 (443,69), 9 (365,153)
4 (119,297), 2 (443,69)
5 (183,246), 4 (119,297)
7 (324,309), 5 (183,246)
6 (647,333), 7 (324,309)
10 (624,331), 6 (647,333)
8 (669,384), 10 (624,331)
Generation 1 Cost 1379 Mutated 20,00%
4 (119,297), 7 (324,309)
3 (33,251), 4 (119,297)
5 (183,246), 3 (33,251)
2 (443,69), 5 (183,246)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
8 (669,384), 9 (365,153)
10 (624,331), 8 (669,384)
6 (647,333), 10 (624,331)
Generation 2 Cost 1643 Mutated 10,00%
7 (324,309), 9 (365,153)
4 (119,297), 7 (324,309)
3 (33,251), 4 (119,297)
5 (183,246), 3 (33,251)
8 (669,384), 5 (183,246)
10 (624,331), 8 (669,384)
6 (647,333), 10 (624,331)
1 (379,78), 6 (647,333)
2 (443,69), 1 (379,78)
Generation 3 Cost 1760 Mutated 2,00%
1 (379,78), 3 (33,251)
9 (365,153), 1 (379,78)
4 (119,297), 9 (365,153)
5 (183,246), 4 (119,297)
7 (324,309), 5 (183,246)
8 (669,384), 7 (324,309)
10 (624,331), 8 (669,384)
6 (647,333), 10 (624,331)
2 (443,69), 6 (647,333)
Generation 4 Cost 1679 Mutated 12,00%
2 (443,69), 1 (379,78)
9 (365,153), 2 (443,69)
3 (33,251), 9 (365,153)
7 (324,309), 3 (33,251)
4 (119,297), 7 (324,309)
5 (183,246), 4 (119,297)
10 (624,331), 5 (183,246)
8 (669,384), 10 (624,331)
6 (647,333), 8 (669,384)
Generation 5 Cost 1550 Mutated 10,00%
6 (647,333), 8 (669,384)
10 (624,331), 6 (647,333)
2 (443,69), 10 (624,331)
```

```
1 (379,78), 2 (443,69)
5 (183,246), 1 (379,78)
7 (324,309), 5 (183,246)
3 (33,251), 7 (324,309)
4 (119,297), 3 (33,251)
9 (365,153), 4 (119,297)
Generation 6 Cost 1662 Mutated 16,00%
6 (647,333), 8 (669,384)
10 (624,331), 6 (647,333)
2 (443,69), 10 (624,331)
1 (379,78), 2 (443,69)
3 (33,251), 1 (379,78)
7 (324,309), 3 (33,251)
5 (183,246), 7 (324,309)
4 (119,297), 5 (183,246)
9 (365,153), 4 (119,297)
Generation 7 Cost 1987 Mutated 12,00%
6 (647,333), 8 (669,384)
10 (624,331), 6 (647,333)
2 (443,69), 10 (624,331)
3 (33,251), 2 (443,69)
1 (379,78), 3 (33,251)
7 (324,309), 1 (379,78)
5 (183,246), 7 (324,309)
4 (119,297), 5 (183,246)
9 (365,153), 4 (119,297)
Generation 8 Cost 1832 Mutated 14,00%
6 (647,333), 9 (365,153)
10 (624,331), 6 (647,333)
8 (669,384), 10 (624,331)
1 (379,78), 8 (669,384)
3 (33,251), 1 (379,78)
4 (119,297), 3 (33,251)
5 (183,246), 4 (119,297)
7 (324,309), 5 (183,246)
2 (443,69), 7 (324,309)
Generation 9 Cost 1754 Mutated 4,00%
1 (379,78), 9 (365,153)
10 (624,331), 1 (379,78)
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
3 (33,251), 8 (669,384)
4 (119,297), 3 (33,251)
5 (183,246), 4 (119,297)
7 (324,309), 5 (183,246)
2 (443,69), 7 (324,309)
Generation 10 Cost 1692 Mutated 8,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
9 (365,153), 8 (669,384)
7 (324,309), 9 (365,153)
4 (119,297), 7 (324,309)
3 (33,251), 4 (119,297)
2 (443,69), 3 (33,251)
1 (379,78), 2 (443,69)
5 (183,246), 1 (379,78)
Generation 11 Cost 1631 Mutated 16,00%
10 (624,331), 9 (365,153)
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
4 (119,297), 7 (324,309)
5 (183,246), 4 (119,297)
3 (33,251), 5 (183,246)
```

```
1 (379,78), 3 (33,251)
2 (443,69), 1 (379,78)
Generation 12 Cost 1505 Mutated 6,00%
8 (669,384), 10 (624,331)
6 (647,333), 8 (669,384)
9 (365,153), 6 (647,333)
7 (324,309), 9 (365,153)
4 (119,297), 7 (324,309)
5 (183,246), 4 (119,297)
3 (33,251), 5 (183,246)
1 (379,78), 3 (33,251)
2 (443,69), 1 (379,78)
Generation 13 Cost 1542 Mutated 8,00%
10 (624,331), 8 (669,384)
6 (647,333), 10 (624,331)
7 (324,309), 6 (647,333)
9 (365,153), 7 (324,309)
4 (119,297), 9 (365,153)
5 (183,246), 4 (119,297)
3 (33,251), 5 (183,246)
1 (379,78), 3 (33,251)
2 (443,69), 1 (379,78)
Generation 14 Cost 1244 Mutated 10,00%
8 (669,384), 10 (624,331)
6 (647,333), 8 (669,384)
7 (324,309), 6 (647,333)
1 (379,78), 7 (324,309)
2 (443,69), 1 (379,78)
9 (365,153), 2 (443,69)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 15 Cost 1244 Mutated 20,00%
8 (669,384), 10 (624,331)
6 (647,333), 8 (669,384)
7 (324,309), 6 (647,333)
1 (379,78), 7 (324,309)
2 (443,69), 1 (379,78)
9 (365,153), 2 (443,69)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 16 Cost 1228 Mutated 12,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
1 (379,78), 7 (324,309)
2 (443,69), 1 (379,78)
9 (365,153), 2 (443,69)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 17 Cost 1228 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
1 (379,78), 7 (324,309)
2 (443,69), 1 (379,78)
9 (365,153), 2 (443,69)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 18 Cost 1224 Mutated 12,00%
6 (647,333), 10 (624,331)
```

```
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
9 (365,153), 7 (324,309)
1 (379,78), 9 (365,153)
2 (443,69), 1 (379,78)
5 (183,246), 2 (443,69)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 19 Cost 1228 Mutated 4,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
1 (379,78), 7 (324,309)
2 (443,69), 1 (379,78)
9 (365,153), 2 (443,69)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 20 Cost 1220 Mutated 4,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 21 Cost 1224 Mutated 14,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
9 (365,153), 7 (324,309)
1 (379,78), 9 (365,153)
2 (443,69), 1 (379,78)
5 (183,246), 2 (443,69)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 22 Cost 1224 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
9 (365,153), 7 (324,309)
1 (379,78), 9 (365,153)
2 (443,69), 1 (379,78)
5 (183,246), 2 (443,69)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 23 Cost 1224 Mutated 16,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
9 (365,153), 7 (324,309)
1 (379,78), 9 (365,153)
2 (443,69), 1 (379,78)
5 (183,246), 2 (443,69)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 24 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
```

```
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 25 Cost 1220 Mutated 4,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 26 Cost 1206 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
9 (365,153), 7 (324,309)
2 (443,69), 9 (365,153)
1 (379,78), 2 (443,69)
5 (183,246), 1 (379,78)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 27 Cost 1220 Mutated 2,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 28 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 29 Cost 1220 Mutated 4,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 30 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
```

```
Generation 31 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 32 Cost 1220 Mutated 16,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 33 Cost 1220 Mutated 2,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 34 Cost 1220 Mutated 14,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 35 Cost 1220 Mutated 20,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 36 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 37 Cost 1220 Mutated 14,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
```

```
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 38 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 39 Cost 1220 Mutated 8,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 40 Cost 1220 Mutated 12,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 41 Cost 1220 Mutated 8,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 42 Cost 1220 Mutated 8,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 43 Cost 1220 Mutated 12,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
```

```
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 44 Cost 1220 Mutated 12,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 45 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 46 Cost 1220 Mutated 12,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 47 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 48 Cost 1220 Mutated 8,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 49 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 50 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
```

```
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 51 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 52 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 53 Cost 1220 Mutated 4,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 54 Cost 1220 Mutated 4,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 55 Cost 1220 Mutated 16,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 56 Cost 1220 Mutated 8,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
```

```
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 57 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 58 Cost 1220 Mutated 14,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 59 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 60 Cost 1220 Mutated 12,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 61 Cost 1220 Mutated 8,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 62 Cost 1220 Mutated 16,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
```

```
Generation 63 Cost 1220 Mutated 14,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 64 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 65 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 66 Cost 1220 Mutated 14,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 67 Cost 1220 Mutated 18,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 68 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 69 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
```

```
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 70 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 71 Cost 1220 Mutated 4,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 72 Cost 1220 Mutated 8,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 73 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 74 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 75 Cost 1220 Mutated 4,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
```

```
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 76 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 77 Cost 1220 Mutated 4,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 78 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 79 Cost 1220 Mutated 16,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 80 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 81 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 82 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
```

```
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 83 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 84 Cost 1220 Mutated 12,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 85 Cost 1220 Mutated 12,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 86 Cost 1220 Mutated 18,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 87 Cost 1220 Mutated 12,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 88 Cost 1220 Mutated 4,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
```

```
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 89 Cost 1220 Mutated 4,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 90 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 91 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 92 Cost 1220 Mutated 16,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 93 Cost 1220 Mutated 8,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 94 Cost 1220 Mutated 14,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
```

```
Generation 95 Cost 1220 Mutated 2,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 96 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 97 Cost 1220 Mutated 12,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 98 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 99 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 100 Cost 1220 Mutated 8,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 101 Cost 1220 Mutated 8,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
```

```
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 102 Cost 1220 Mutated 4,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 103 Cost 1220 Mutated 12,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 104 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 105 Cost 1220 Mutated 8,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 106 Cost 1220 Mutated 8,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 107 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
```

```
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 108 Cost 1220 Mutated 8,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 109 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 110 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 111 Cost 1220 Mutated 8,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 112 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 113 Cost 1220 Mutated 12,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 114 Cost 1220 Mutated 2,00%
6 (647,333), 10 (624,331)
```

```
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 115 Cost 1220 Mutated 8,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 116 Cost 1220 Mutated 4,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 117 Cost 1220 Mutated 8,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 118 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 119 Cost 1220 Mutated 18,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 120 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
```

```
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 121 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 122 Cost 1220 Mutated 12,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 123 Cost 1220 Mutated 12,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 124 Cost 1220 Mutated 10,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 125 Cost 1220 Mutated 8,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Generation 126 Cost 1220 Mutated 8,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
```

```
Generation 127 Cost 1220 Mutated 6,00%
6 (647,333), 10 (624,331)
8 (669,384), 6 (647,333)
7 (324,309), 8 (669,384)
2 (443,69), 7 (324,309)
1 (379,78), 2 (443,69)
9 (365,153), 1 (379,78)
5 (183,246), 9 (365,153)
4 (119,297), 5 (183,246)
3 (33,251), 4 (119,297)
Solution found after 127 generations.
```