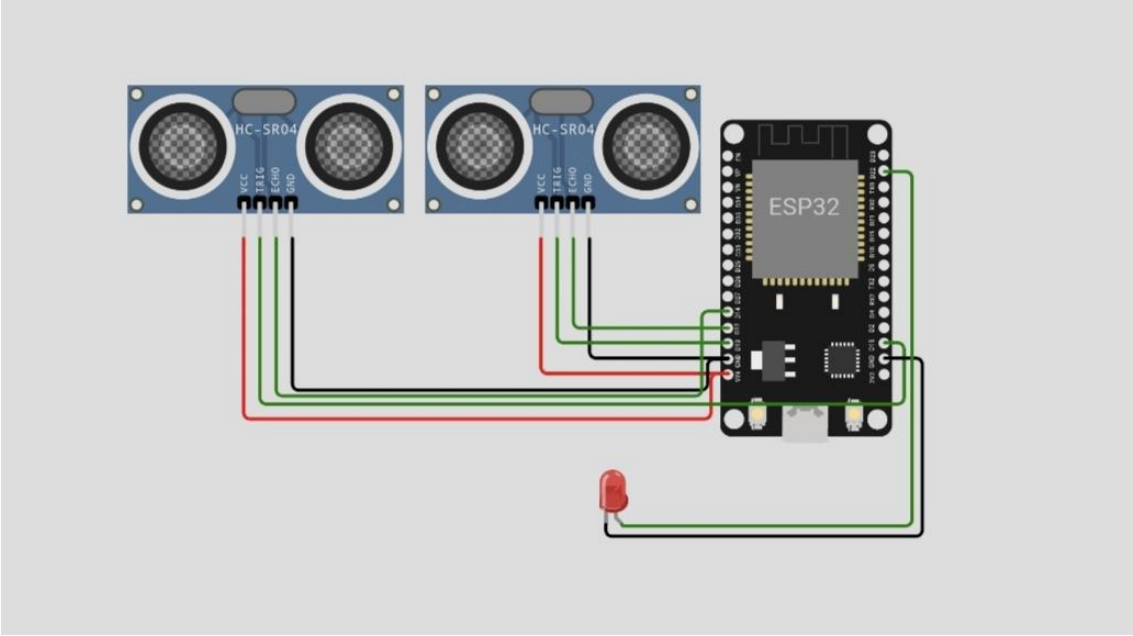
# DEVELOPING PUBLIC TRANSPORT OPTIMIZATION

## IOT BASED PUBLIC TRANSPORT OPTIMIZATION USING ARDUINO:

IoT(Internet of Things) can assist in integration of communication, control and information processing across various transportation systems. In public transportation, there is lack of real time information. The public transit usage can be improved if real time information of the vehicle such as the seating availability, current location and time taken to reach the destination are provided with easier access. It would also be helpful for the passengers to find alternate choices depending on their circumstances. As excessive long waiting often discourage the travelers and makes them reluctant to take buses. A smart information system has been proposed where the travelers get prior information about current location, next location of bus and crowd level inside the bus. This system is designed using ARDUINO UNO, IR Sensor and GPS Module. An Intelligent Transport System (ITS) removes the barriers for public transport usage and creates the positive impact about the bus journey.

### Circuit diagram:




### COMPONENTS REQUIRED:

- HC SR04
- ESP32

- Connecting wires
- LED

**CODE:**


```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": 15,
   "metadata": {
    "id": "R4ag3SOp2uAM"
   },
   "outputs": [],
   "source": [
    "import numpy as np, random, operator, pandas as pd, matplotlib.pyplot as plt"
   ]
  },
  {
   "cell_type": "code",
   "source": [
    "city_name_data = pd.read_csv('sgb128_name.csv',header=None)\n",
    "city_dist_data = pd.read_csv('sgb128_dist.csv',header=None)\n",
    "city_weight_data = pd.read_csv('sgb128_weight.csv',header=None)\n",
    "# city_length = 25"
   ],
   "metadata": {
    "id": "YYzhm4cjUVk4"
```

```
   },
   "execution_count": 16,
   "outputs": []
  },
  {
   "cell_type": "code",
   "execution_count": 17,
   "metadata": {
    "id": "f40wgYcI2yTr"
   },
   "outputs": [],
   "source": [
    "class City:\n",
    "    def __init__(self,name,population):\n",
    "        self.name=name\n",
    "        self.population=population\n",
    "    \n",
    "    def distance(self, city):\n",
    "        distance=city_dist_data.iloc[self.name,city.name]\n",
    "        return distance\n",
    "    \n",
    "    def __repr__(self):\n",
    "        return \"\\\\\"+str(city_name_data.iloc[self.name,0])
+\"[\"+str(self.population)+\"]\"+\"\\\\\""
   ]
  },
  {
   "cell_type": "code",
```

```
"execution_count": 18,
"metadata": {
  "id": "OizIyOa0265B"
},
"outputs": [],
"source": [
  "class Fitness:\n",
  "    def __init__(self, route):\n",
  "        self.route = route\n",
  "        self.distance = 0\n",
  "        self.fitness= 0.0\n",
  "        self.total_population= 0\n",
  "    \n",
  "    def routeDistance(self):\n",
  "        pathDistance = 0\n",
  "        for i in range(0, len(self.route)):\n",
  "            fromCity = self.route[i]\n",
  "            toCity = None\n",
  "            if i + 1 < len(self.route):\n",
  "                toCity = self.route[i + 1]\n",
  "            else:\n",
  "                break\n",
  "            pathDistance += fromCity.distance(toCity)\n",
  "        self.distance = pathDistance\n",
  "        return self.distance\n",
  "    \n",
  "    def routePopulation(self):\n",
```

```
    "        path_population = 0\n",
    "        for i in range(0, len(self.route)):\n",
    "            City = self.route[i]\n",
    "            path_population += City.population\n",
    "        self.total_population = path_population\n",
    "        return self.total_population\n",
    "    \n",
    "    def routeFitness(self):\n",
    "        if self.fitness == 0:\n",
    "          self.fitness = self.routePopulation() / float(self.routeDistance()) \n",
    "        return self.fitness"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 19,
   "metadata": {
    "id": "K4uLoioc3FYE"
   },
   "outputs": [],
   "source": [
    "def createRoute(cityList):\n",
    "    route = random.sample(cityList, 10)\n",
    "    return route\n",
    "    \n",
    "def initialPopulation(popSize, cityList):\n",
    "    population = []\n",
```

```
    "    for i in range(0, popSize):\n",
    "        population.append(createRoute(cityList))\n",
    "    return population"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 20,
   "metadata": {
    "id": "N2F1Ei5O3KCR"
   },
   "outputs": [],
   "source": [
    "def rankRoutes(population):\n",
    "    fitnessResults = {}\n",
    "    for i in range(0,len(population)):\n",
    "        fitnessResults[i] = Fitness(population[i]).routeFitness()\n",
    "    \n",
    "    return sorted(fitnessResults.items(), key = operator.itemgetter(1), reverse = True)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 21,
   "metadata": {
    "id": "GvHkFsLx3Nzo"
   },
```

```
 "outputs": [],
 "source": [
  "def selection(popRanked, eliteSize):\n",
  "    selectionResults = []\n",
  "    df = pd.DataFrame(np.array(popRanked), columns=[\"Index\",\"Fitness\"])\n",
  "    df['cum_sum'] = df.Fitness.cumsum()\n",
  "    df['cum_perc'] = 100*df.cum_sum/df.Fitness.sum()\n",
  "    \n",
  "    for i in range(0, eliteSize):\n",
  "        selectionResults.append(popRanked[i][0])\n",
  "    for i in range(0, len(popRanked) - eliteSize):\n",
  "        pick = 100*random.random()\n",
  "        for i in range(0, len(popRanked)):\n",
  "            if pick <= df.iat[i,3]:\n",
  "                selectionResults.append(popRanked[i][0])\n",
  "                break\n",
  "    return selectionResults"
 ]
},
{
 "cell_type": "code",
 "execution_count": 22,
 "metadata": {
  "id": "DIPlzfSc3RKe"
 },
 "outputs": [],
 "source": [
```

```
   "def matingPool(population, selectionResults):\n",
   "    matingpool = []\n",
   "    for i in range(0, len(selectionResults)):\n",
   "        index = selectionResults[i]\n",
   "        matingpool.append(population[index])\n",
   "    return matingpool"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 23,
  "metadata": {
   "id": "JMVLhepw3bHn"
  },
  "outputs": [],
  "source": [
   "def breed(parent1, parent2):\n",
   "    child = []\n",
   "    childP1 = []\n",
   "    childP2 = []\n",
   "    \n",
   "    geneA = int(random.random() * len(parent1))\n",
   "    geneB = int(random.random() * len(parent1))\n",
   "    \n",
   "    startGene = min(geneA, geneB)\n",
   "    endGene = max(geneA, geneB)\n",
   "\n",
```

```
    "    for i in range(startGene, endGene):\n",
    "        childP1.append(parent1[i])\n",
    "    for item in parent2:\n",
    "      if item not in childP1 and len(childP1)<10:\n",
    "        childP1.append(item)\n",
    "    child = childP1\n",
    "    return child\n",
    "\n",
    "def breedPopulation(matingpool, eliteSize):\n",
    "    children = []\n",
    "    length = len(matingpool) - eliteSize\n",
    "    pool = random.sample(matingpool, len(matingpool))\n",
    "\n",
    "    for i in range(0,eliteSize):\n",
    "        children.append(matingpool[i])\n",
    "    \n",
    "    for i in range(0, length):\n",
    "        child = breed(pool[i], pool[len(matingpool)-i-1])\n",
    "        children.append(child)\n",
    "    return children"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 24,
   "metadata": {
    "id": "Wq1SM1Hg3c7c"
```

```
 },
 "outputs": [],
 "source": [
  "def mutate(individual, mutationRate):\n",
  "    for swapped in range(len(individual)):\n",
  "        if(random.random() < mutationRate):\n",
  "            swapWith = int(random.random() * len(individual))\n",
  "            \n",
  "            city1 = individual[swapped]\n",
  "            city2 = individual[swapWith]\n",
  "            \n",
  "            individual[swapped] = city2\n",
  "            individual[swapWith] = city1\n",
  "    return individual\n",
  "    \n",
  "def mutatePopulation(population, mutationRate):\n",
  "    mutatedPop = []\n",
  "    \n",
  "    for ind in range(0, len(population)):\n",
  "        mutatedInd = mutate(population[ind], mutationRate)\n",
  "        mutatedPop.append(mutatedInd)\n",
  "    return mutatedPop"
 ]
},
{
 "cell_type": "code",
 "execution_count": 25,
```

```
   "metadata": {
    "id": "OzUaK9kK3kdo"
   },
   "outputs": [],
   "source": [
    "def nextGeneration(currentGen, eliteSize, mutationRate):\n",
    "    popRanked = rankRoutes(currentGen)\n",
    "    selectionResults = selection(popRanked, eliteSize)\n",
    "    matingpool = matingPool(currentGen, selectionResults)\n",
    "    children = breedPopulation(matingpool, eliteSize)\n",
    "    nextGeneration = mutatePopulation(children, mutationRate)\n",
    "    return nextGeneration"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 26,
   "metadata": {
    "id": "Zg77N1J13nn2"
   },
   "outputs": [],
   "source": [
    "def geneticAlgorithm(population, popSize, eliteSize, mutationRate, generations):\n",
    "    pop = initialPopulation(popSize, population)\n",
    "    print(\"Initial distance: \" + str(Fitness(pop[rankRoutes(pop)[0][0]]).routeDistance()))\n",
    "    print(\"Initial population: \" +
str(Fitness(pop[rankRoutes(pop)[0][0]]).routePopulation()))\n",
    "    \n",
```

```
    "    for i in range(0, generations):\n",
    "        pop = nextGeneration(pop, eliteSize, mutationRate)\n",
    "\n",
    "    total_distance=0\n",
    "    total_population=0\n",
    "    for i in range (0,10):\n",
    "        bestRouteIndex = rankRoutes(pop)[i][0]\n",
    "        bestRoute = pop[bestRouteIndex]\n",
    "        fitness = Fitness(bestRoute)\n",
    "        fitness.routeFitness()\n",
    "        print((i+1),\"route:\", bestRoute)\n",
    "        total_distance += fitness.distance\n",
    "        print(\"distance: \" + str(fitness.distance))\n",
    "        total_population += fitness.total_population\n",
    "        print(\"Population: \" + str(fitness.total_population))\n",
    "    \n",
    "    print(\"Total distance= \"+ str(total_distance))\n",
    "    print(\"Total population= \" + str(total_population))\n",
    "    return bestRoute"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 27,
   "metadata": {
    "id": "xGtP1FF-3qYh"
   },
```

```
    "outputs": [],
    "source": [
     "# cityList = []\n",
     "# len(cities.City)\n",
     "# for i in range(0,len(cities.City)):\n",
"cityList = []\n",
     "for i in range(0,len(city_name_data)):\n",
     "    cityList.append(City(name=i,population=city_weight_data.iloc[i,0]))\n"
    ]
   },
   {
    "cell_type": "code",
    "execution_count": 28,
    "metadata": {
     "colab": {
      "base_uri": "https://localhost:8080/"
     },
     "id": "ycdq0fIW3sqN",
     "outputId": "e6d7d2da-5aed-4d5f-f1c7-eb0ae9e82d88"
    },
    "outputs": [
     {
      "output_type": "stream",
      "name": "stdout",
      "text": [
       "Initial distance: 8587\n",
       "Initial population: 3114379\n",
```

"1 route: [\"Saint Louis, MO[453085]\", \"San Jose, CA[629546]\", \"Worcester, MA[161799]\", \"San Diego, CA[875538]\", \"Ravenna, OH[11987]\", \"Winnipeg, MB[564473]\", \"Rochester, NY[241741]\", \"Toledo, OH[354635]\", \"Rockford, IL[139712]\", \"San Francisco, CA[678974]\"]\n",

"distance: 3693\n",

"Population: 4111490\n",

"2 route: [\"Washington, DC[638432]\", \"Winnipeg, MB[564473]\", \"Rochester, NY[241741]\", \"Saint Joseph, MO[76691]\", \"San Antonio, TX[786023]\", \"Rockford, IL[139712]\", \"Toledo, OH[354635]\", \"San Jose, CA[629546]\", \"Worcester, MA[161799]\", \"San Diego, CA[875538]\"]\n",

"distance: 4098\n",

"Population: 4468590\n",

"3 route: [\"Toronto, ON[599217]\", \"Rockford, IL[139712]\", \"San Antonio, TX[786023]\", \"Saint Louis, MO[453085]\", \"Worcester, MA[161799]\", \"San Jose, CA[629546]\", \"Toledo, OH[354635]\", \"Rochester, NY[241741]\", \"Seattle, WA[493846]\", \"San Diego, CA[875538]\"]\n",

"distance: 4527\n",

"Population: 4735142\n",

"4 route: [\"Seattle, WA[493846]\", \"Saint Louis, MO[453085]\", \"San Jose, CA[629546]\", \"Worcester, MA[161799]\", \"San Diego, CA[875538]\", \"Rockford, IL[139712]\", \"Ravenna, OH[11987]\", \"San Antonio, TX[786023]\", \"Rochester, NY[241741]\", \"Toledo, OH[354635]\"]\n",

"distance: 4006\n",

"Population: 4147912\n",

"5 route: [\"Saint Louis, MO[453085]\", \"Worcester, MA[161799]\", \"San Diego, CA[875538]\", \"Winnipeg, MB[564473]\", \"Rochester, NY[241741]\", \"San Antonio, TX[786023]\", \"Toronto, ON[599217]\", \"Rockford, IL[139712]\", \"Ravenna, OH[11987]\", \"San Jose, CA[629546]\"]\n",

"distance: 4342\n",

"Population: 4463121\n",

"6 route: [\"San Jose, CA[629546]\", \"Rochester, NY[241741]\", \"San Diego, CA[875538]\", \"San Antonio, TX[786023]\", \"Rockford, IL[139712]\", \"Toronto, ON[599217]\", \"Ravenna, OH[11987]\", \"Saint Louis, MO[453085]\", \"Winnipeg, MB[564473]\", \"Washington, DC[638432]\"]\n",

"distance: 4846\n",

"Population: 4939754\n",

"7 route: [\"San Jose, CA[629546]\", \"Toledo, OH[354635]\", \"Rochester, NY[241741]\", \"San Antonio, TX[786023]\", \"Washington, DC[638432]\", \"Toronto, ON[599217]\", \"Ravenna, OH[11987]\", \"Winnipeg, MB[564473]\", \"San Diego, CA[875538]\", \"Worcester, MA[161799]\"]\n",

"distance: 4788\n",

"Population: 4863391\n",

"8 route: [\"Washington, DC[638432]\", \"Winnipeg, MB[564473]\", \"Rochester, NY[241741]\", \"San Antonio, TX[786023]\", \"Toledo, OH[354635]\", \"Rockford, IL[139712]\", \"Saint Joseph, MO[76691]\", \"San Jose, CA[629546]\", \"Worcester, MA[161799]\", \"San Diego, CA[875538]\"]\n",

"distance: 4540\n",

"Population: 4468590\n",

"9 route: [\"Worcester, MA[161799]\", \"San Jose, CA[629546]\", \"Rochester, NY[241741]\", \"San Diego, CA[875538]\", \"Toronto, ON[599217]\", \"Ravenna, OH[11987]\", \"Saint Louis, MO[453085]\", \"Saint Joseph, MO[76691]\", \"San Francisco, CA[678974]\", \"Rockford, IL[139712]\"]\n",

"distance: 4031\n",

"Population: 3868290\n",

"10 route: [\"San Diego, CA[875538]\", \"Worcester, MA[161799]\", \"San Jose, CA[629546]\", \"Seattle, WA[493846]\", \"Saint Louis, MO[453085]\", \"Toledo, OH[354635]\", \"Rochester, NY[241741]\", \"San Antonio, TX[786023]\", \"Rockford, IL[139712]\", \"Ravenna, OH[11987]\"]\n",

"distance: 4424\n",

"Population: 4147912\n",

"Total distance= 43295\n",

"Total population= 44214192\n"

]

},

{

"output_type": "execute_result",

"data": {

"text/plain": [

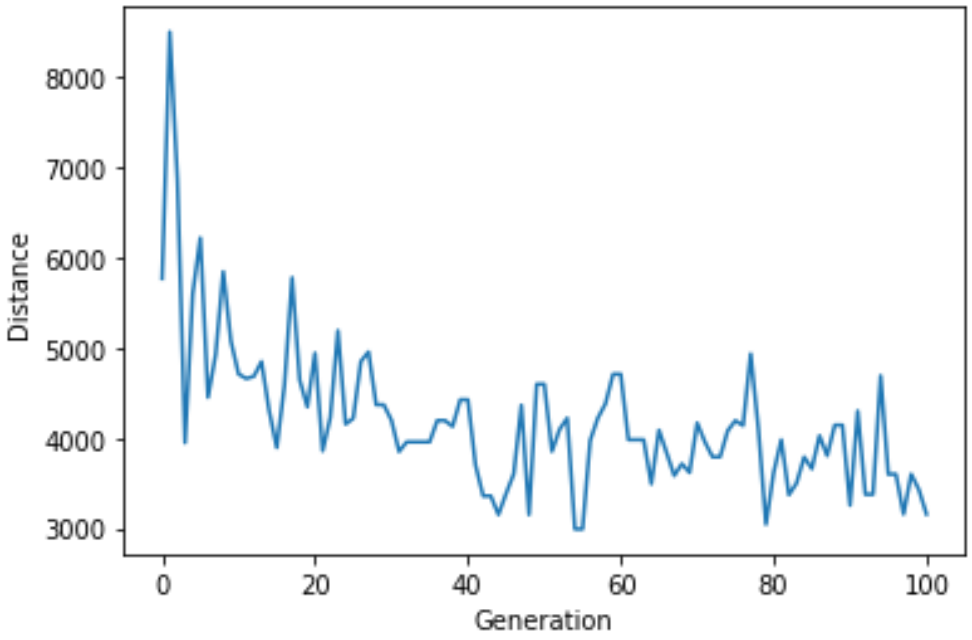
```
            "[\"San Diego, CA[875538]\",\n",
            " \"Worcester, MA[161799]\",\n",
            " \"San Jose, CA[629546]\",\n",
            " \"Seattle, WA[493846]\",\n",
            " \"Saint Louis, MO[453085]\",\n",
            " \"Toledo, OH[354635]\",\n",
            " \"Rochester, NY[241741]\",\n",
            " \"San Antonio, TX[786023]\",\n",
            " \"Rockford, IL[139712]\",\n",
            " \"Ravenna, OH[11987]\"]"
          ]
         },
         "metadata": {},
         "execution_count": 28
       }
      ],
      "source": [
       "\n",
       "geneticAlgorithm(population=cityList, popSize=600, eliteSize=20, mutationRate=0.15,
generations=100)\n"
      ]
     },
     {
      "cell_type": "code",
      "execution_count": 29,
      "metadata": {
       "id": "R8XUydKH3uXX"
      },
```

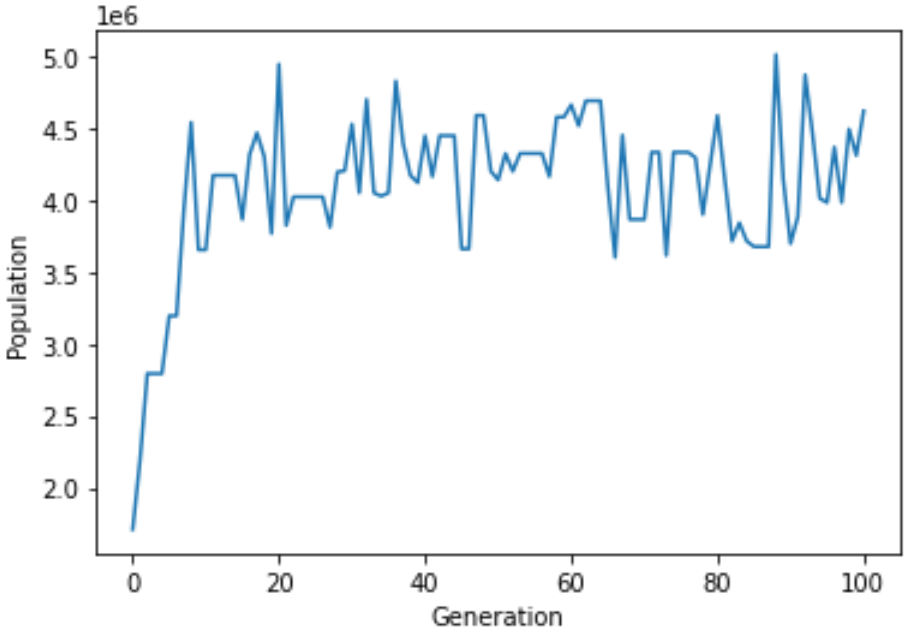
```
"outputs": [],

"source": [

 "def geneticAlgorithmDistancePlot(population, popSize, eliteSize, mutationRate, generations):\n",

 "    pop = initialPopulation(popSize, population)\n",

 "    progress = []\n",

 "    progress.append(Fitness(pop[rankRoutes(pop)[0][0]]).routeDistance())\n",

 "    \n",

 "    for i in range(0, generations):\n",

 "        pop = nextGeneration(pop, eliteSize, mutationRate)\n",

 "        progress.append(Fitness(pop[rankRoutes(pop)[0][0]]).routeDistance())\n",

 "    \n",

 "    plt.plot(progress)\n",

 "    plt.ylabel('Distance')\n",

 "    plt.xlabel('Generation')\n",

 "    plt.show()\n",

 "def geneticAlgorithmPopulationPlot(population, popSize, eliteSize, mutationRate, generations):\n",

 "    pop = initialPopulation(popSize, population)\n",

 "    progress = []\n",

 "    progress.append(Fitness(pop[rankRoutes(pop)[0][0]]).routePopulation())\n",

 "    \n",

 "    for i in range(0, generations):\n",

 "        pop = nextGeneration(pop, eliteSize, mutationRate)\n",

 "        progress.append(Fitness(pop[rankRoutes(pop)[0][0]]).routePopulation())\n",

 "    \n",

 "    plt.plot(progress)\n",

 "    plt.ylabel('Population')\n",
```

```
    "    plt.xlabel('Generation')\n",
    "    plt.show()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 31,
   "metadata": {
    "id": "jD_0FY4bh9bF",
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 552
    },
    "outputId": "33c134bc-72eb-4ea0-b557-aa60f48255e5"
   },
   "outputs": [
    {
     "output_type": "display_data",
     "data": {
      "text/plain": [
       "<Figure size 432x288 with 1 Axes>"
      ],
     },
     "metadata": {
      "needs_background": "light"
     }
    }
```

```
      ],
      "source": [
       "\n",
       "geneticAlgorithmDistancePlot(population=cityList, popSize=600, eliteSize=80, mutationRate=0.15, generations=100)\n",
       "geneticAlgorithmPopulationPlot(population=cityList, popSize=600, eliteSize=80, mutationRate=0.15, generations=100)\n"
      ]
    }
  ],
  "metadata": {
    "colab": {
      "name": "Route optimization with genetic approach.ipynb",
      "provenance": [],
      "collapsed_sections": []
    },
    "kernelspec": {
      "display_name": "Python 3",
      "name": "python3"
    },
    "language_info": {
      "name": "python"
    }
  },
  "nbformat": 4,
  "nbformat_minor": 0
}
```

## Output:




**Conclusion :**

**In this paper a smart information system is presented for the bus passengers that have the ability to interconnect passengers with real-world public bus. The Smart information system based on distributed IoT System consists of IR Sensors, GPS Module, Arduino UNO, and IoT Module to count the number of passengers and to provide information about current and next location of the bus. Since the users are provided with real time information about the vacant seats, the passengers will able to take better decisions in terms of bus.**

## Thankyou!