**SHORTEST PATH PROBLEM WITH GENETIC ALGORITHM**

**Onur Erkin Sucu**

**TABLE OF CONTENTS**

[**INTRODUCTION**](#_72cr3iajq0op) **3**

[**GA FOR SHORTEST PATH PROBLEM**](#_2k0zk5m4o62w) **4**

[Initial Population:](#_hvmon6elitx8) 4

[Objective Function:](#_3guksoeyvvu) 6

[Natural Selection Function(natsec):](#_ltox3pkeu2fz) 8

[Crossover Function](#_qsscomnb7pc1) 10

[Mutation Function:](#_ave8c1iwu0sh) 11

[Repair Function](#_iedgaf8l9vav) 13

[Main Function](#_c1v7ge1axg7g) 14

[**CONCLUSION**](#_h3wt6mw3uwmu) **14**

[**APPENDIX**](#_c4routwc1p0a) **15**

## **INTRODUCTION**

There are several algorithms that can be used to solve shortest path problem, two of them are Dijkstra’s algorithm and genetic algorithm. Both of them have some advantages and disadvantages. In Dijkstra’s algorithm, the solution is the best solution for sure since it consider all the possibilities and finds the best solution. On the other hand, genetic algorithm(GA) may not find the best solution for some cases. However, GA is working faster than Dijkstra’s algorithm and it takes significantly less time to come up with the solution. In this report GA approach to the shortest path problem will be explained.

Genetic algorithms(GAs) are global search and optimization techniques that is inspired by natural selection, evolution and genetics. The genetic algorithms use several operators to create a similar process.

We have used 3 main operators in the code which are “natsec” (natural selection) , “crossover”, “mutation”. These operators will be explained in details in section 2. Also there are 3 other functions that are used in our algorithm namely “createpop” , “objective” and “repair”. These functions also will be explained in details in section 2.



Figure 1: Block Diagram of GA

## **GA FOR SHORTEST PATH PROBLEM**

### Initial Population:

At the beginning of the algorithm we should create an initial population, for this purpose we have develop a function namely “createpop” which creates a population by taking into consideration starting and destination points. Also created chromosomes do not contain any loops.

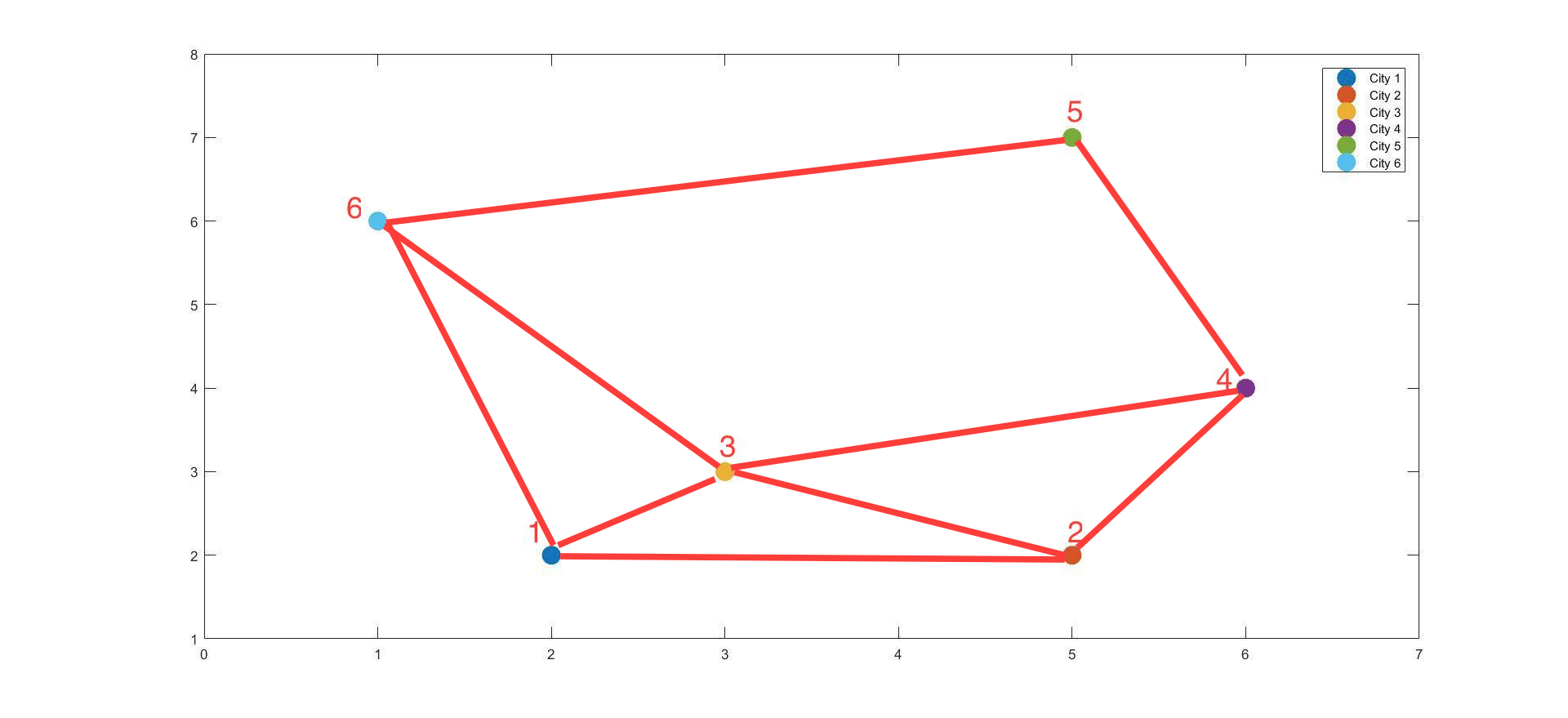


Figure 2. Used city network

For this purpose we created a matrix “m” which contains the network information. In other words, this matrix shows us which cities are connected to each other and which are not.

m =

0 1 1 0 0 1

2 0 2 2 0 0

3 3 0 3 0 3

0 4 4 0 4 0

0 0 0 5 0 5

6 0 6 0 6 0

In this matrix, each row represents a city and each column represents which cities are connected to this city.

Example:

Row 4 is [0 4 4 0 4 0] non zero elements are 2nd 3rd and 5th elements, which means that city 4 is connected with city 2, city 3 and city 5.

Population creator function creates #psize(population size) of chromosomes all of them starts with starting point and ends with destination point. It creates these chromosomes one by one and assign the elements to each chromosome one by one.

Example:

Let's say our starting city is 2 and destination point is 6, and psize is 4. Then our function will start with creating 1st population and assign 2 as first element of first row. Then it randomly goes to a city which is connected with city 2. Let’s say it goes to city 1, then function updates the m matrix by replacing each elements of row 2 with zero. This is because we don’t want any loops in our solutions. By replacing row 2 with zero we are sure that city 2 will not be visited again in our solution.

After going city 2 to city 1 new m matrix is like that:

m =

0 1 1 0 0 1

**0 0 0 0 0 0**

3 3 0 3 0 3

0 4 4 0 4 0

0 0 0 5 0 5

6 0 6 0 6 0

Now we are at city 1 there are two options that we can go city 3 and city 6.

Let’s say we go to city 6 in other words we have reached our destination point. Then, first chromosome of our population is:

2 1 6 0 0 0 0 0 0 0 0 0

This procedure is done psize times to create psize number of chromosomes. An example of population is below.

population =

2 1 6 0 0 0 0 0 0 0 0 0

2 4 3 1 6 0 0 0 0 0 0 0

2 4 5 6 0 0 0 0 0 0 0 0

2 1 3 4 5 6 0 0 0 0 0 0

Our population has 12 columns (twice as city number), the reasons why we have 2\*city number of columns instead of #city columns is that we will assign new elements to the chromosomes in crossover and mutation operators and the new solutions may have more than #city elements in it.



Figure 3.MATLAB code of the createpop function

### Objective Function:

Objective functions calculates the distance travelled for each chromosome. Since we have #psize of chromosomes in our population, it creates a (psize x 1) matrix namely “obj”. It takes the coordinates of each cities and the population information to calculate the distance travelled.

An example of obj matrix for the example population is:

population =

2 1 6 0 0 0 0 0 0 0 0 0

2 4 3 1 6 0 0 0 0 0 0 0

2 4 5 6 0 0 0 0 0 0 0 0

2 1 3 4 5 6 0 0 0 0 0 0

obj=

7.1231

10.9357

9.5215

14.8619

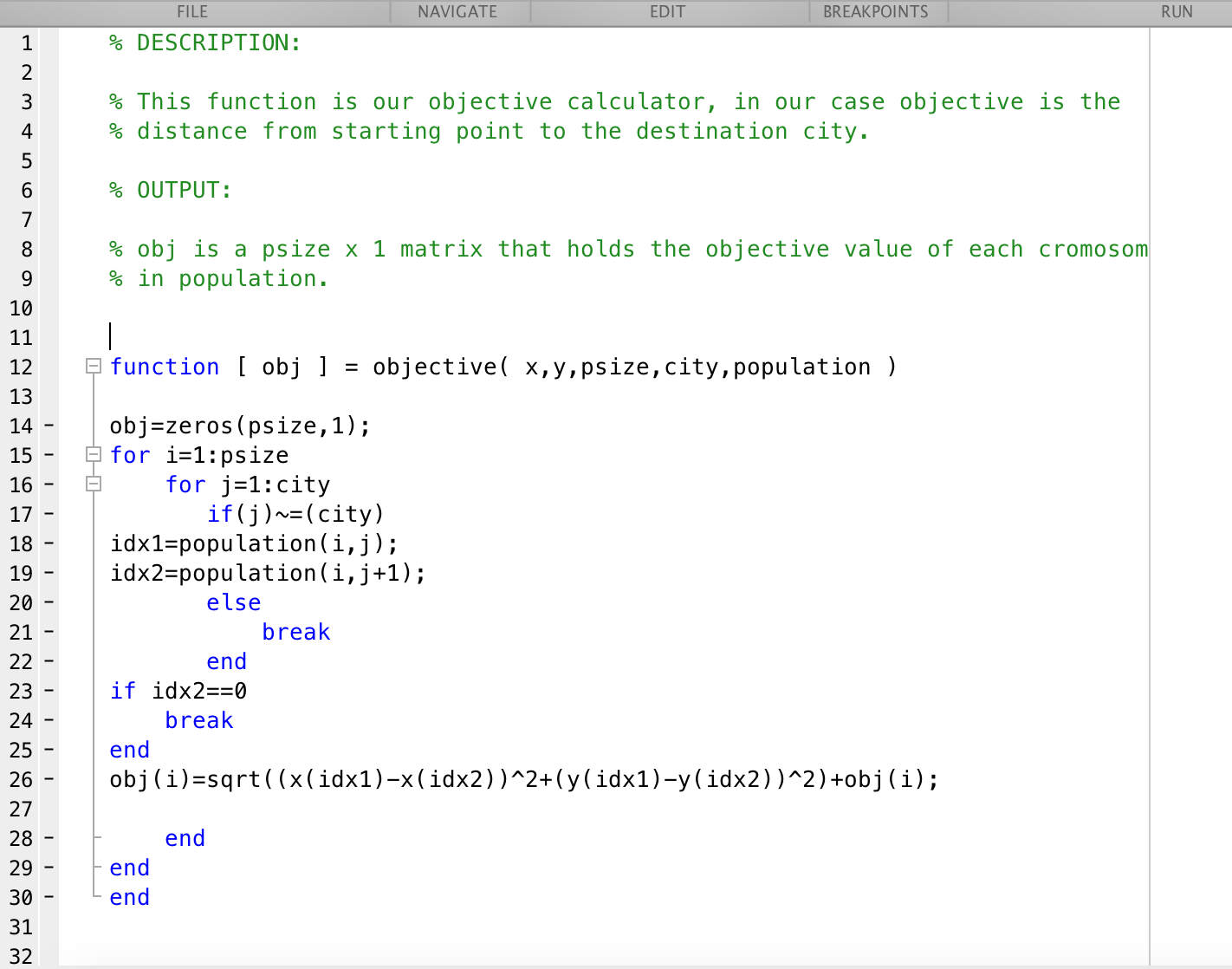


Figure 4. MATLAB code of objective function.

### Natural Selection Function(natsec):

In natural selection function(natsec) we have used roulette wheel type of natural selection. For this purpose first we get the reciprocal of each element in obj matrix, then we divide each element in this matrix to the sum of them to find the probability of each element in obj matrix. Then we get create a matrix to find cumulative probability.

Example:

Probability matrix for the population above is:

probs =

0.3474

0.2263

0.2599

0.1665

And cumulative probability matrix(cprobs) is:

cprobs =

0.3474

0.5736

0.8335

1.0000

Then we randomly create a number between 0 and 1 for every chromosome. The first cprobs element that is bigger than this random number survives. This procedure is repeated #psize times. And we get an intermediate population like this:

Population =

2 1 6 0 0 0 0 0 0 0 0 0

2 4 3 1 6 0 0 0 0 0 0 0

2 4 5 6 0 0 0 0 0 0 0 0

2 1 3 4 5 6 0 0 0 0 0 0

Intermediate Population =

2 1 6 0 0 0 0 0 0 0 0 0

2 4 5 6 0 0 0 0 0 0 0 0

2 1 6 0 0 0 0 0 0 0 0 0

2 4 5 6 0 0 0 0 0 0 0 0

First chromosome appears twice in our intermediate population, this is something we expect because its probability of survival is the highest among all of them with %34.74. We can see that chromosome 2 and 4 is eliminated. This is also something expected because their survival rate is lower than the other two chromosomes.

Basically in GA natural selection operator is very similar to the natural selection in the nature. It depends on the fact that strong ones live and the weak ones die.

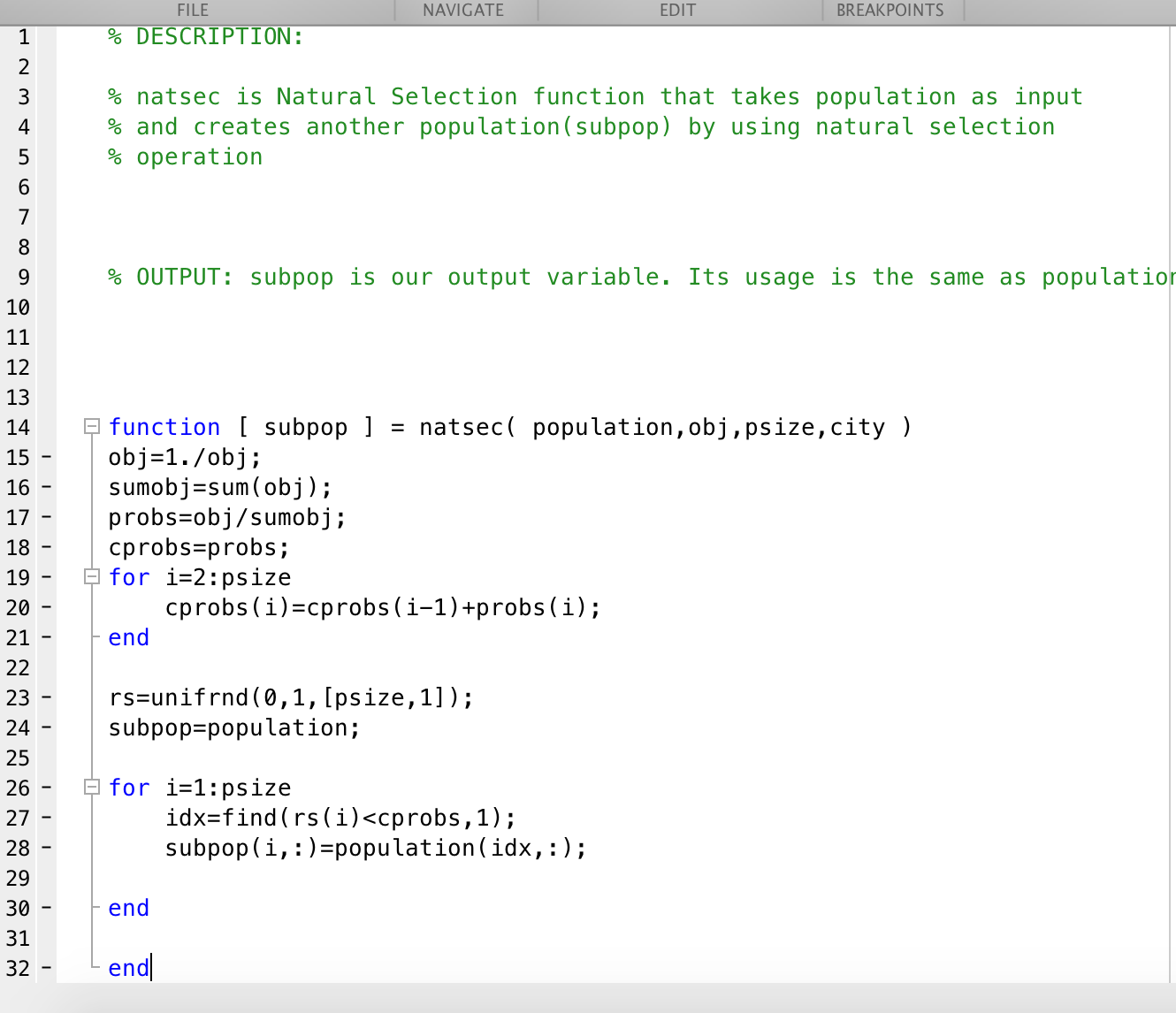


Figure 5. MATLAB code of natsec function.

### Crossover Function

Crossover function is used to increase diversity in the chromosomes. For this purpose one point crossover method which is shown in figure 6 is used. According to the method, two parents are chosen randomly and crossover will be applied to them. Then, if parents have a common gene, crossover will be occurred. Parent1 and parent2 will interchange their genes after the common gene. New chromosomes are named as children1 and children2.

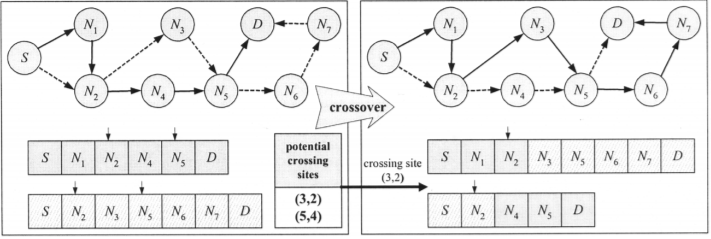


Figure 6. One point crossover diagram.

The code of crossover function is given in figure 7.



Figure 7. MATLAB code of crossover function.

An example for the crossover function is given below. First row of ”population” is parent1 and second row is parent2 in the figure 8, then we obtain children1 and children2 as shown in figure 9.

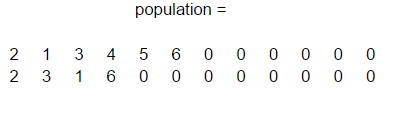


Figure 8. The rows of initial population namely parent1 and parent2.

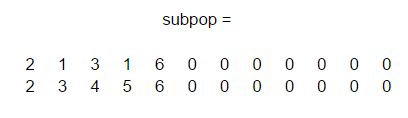


Figure 9. The rows of resultant population namely children1 and children2.

### Mutation Function:

In this function, it is aimed to change chromosome from a specific point which determined randomly. Since there is limited connections between cities, after changing point, connections of cities should be considered. When there is only 2 cities in the chromosome, mutation function cannot be applied. The code of mutation function is given in the figure 10.

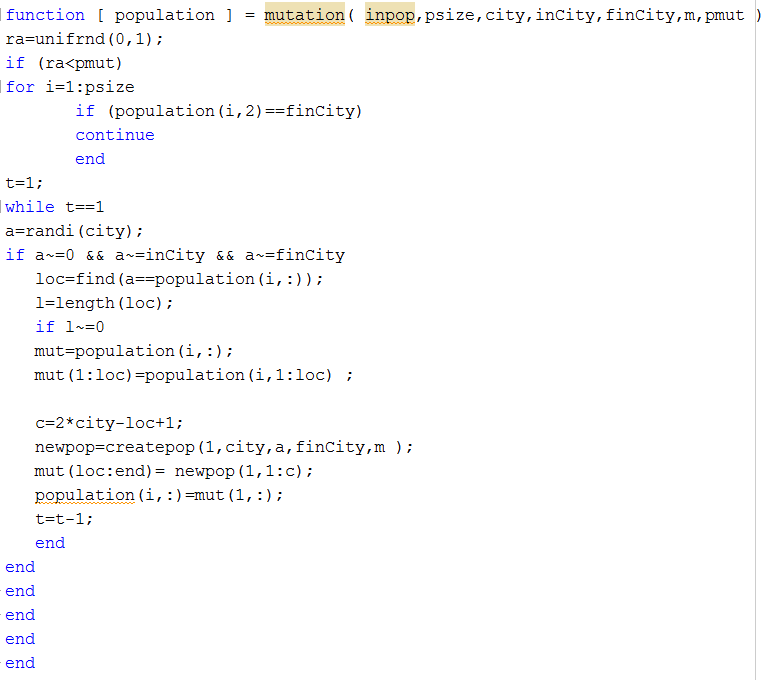
****

Figure 10. MATLAB Code of mutation function.

An example of function over a population is given in figure 11. “population” is generated by applying mutation function to “inpop” population.

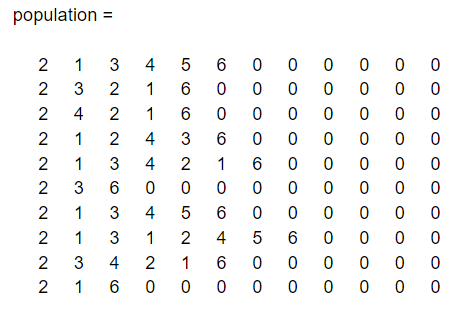
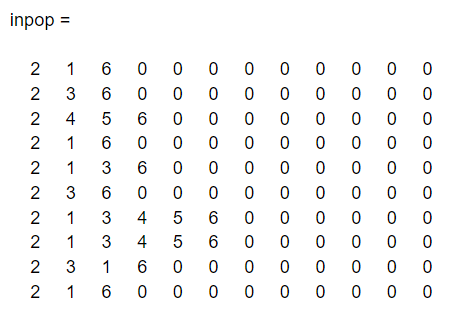
****

Figure 11: “inpop” initial population and the population after mutation function applied.

### Repair Function

We have observed that after crossover and mutation functions, some of chromosomes contain multiple number of same cities.In that case to repair those chromosomes we have implemented a repair function. In this function, we search for multiple number of same cities, than we can eliminate the genes in between repetitive cities in other words we get rid of the loop.

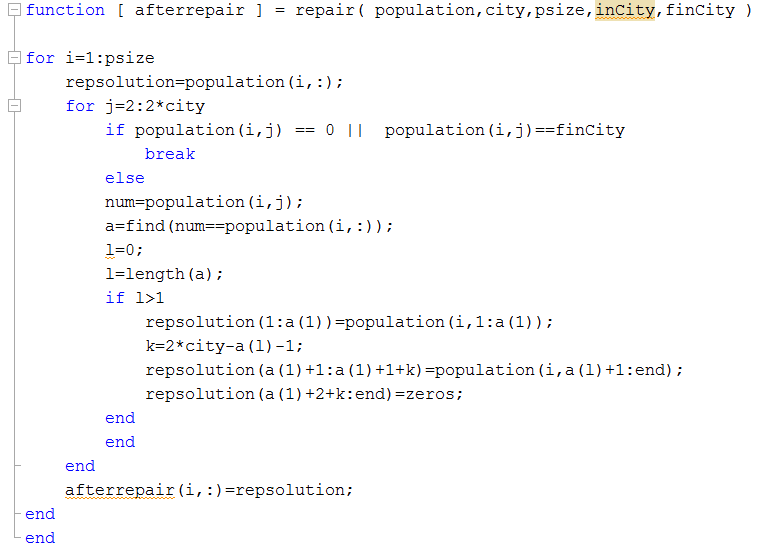
****

Figure 12. Code of Repair Function

In the figure 12, “population” contains some chromosomes that needed to be repaired and “afterrepair” population is the resultant population after repair function is applied to initial population.

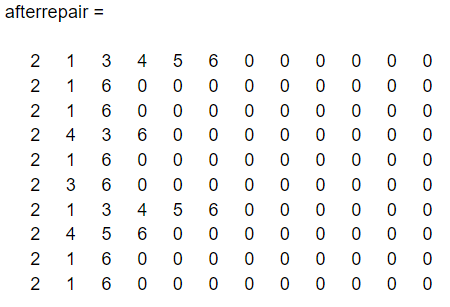
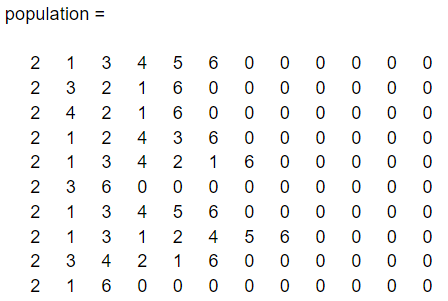


Figure 13. The initial population and the “afterrepaired” population.

### Main Function

Main function contains our other functions and the iteration number.

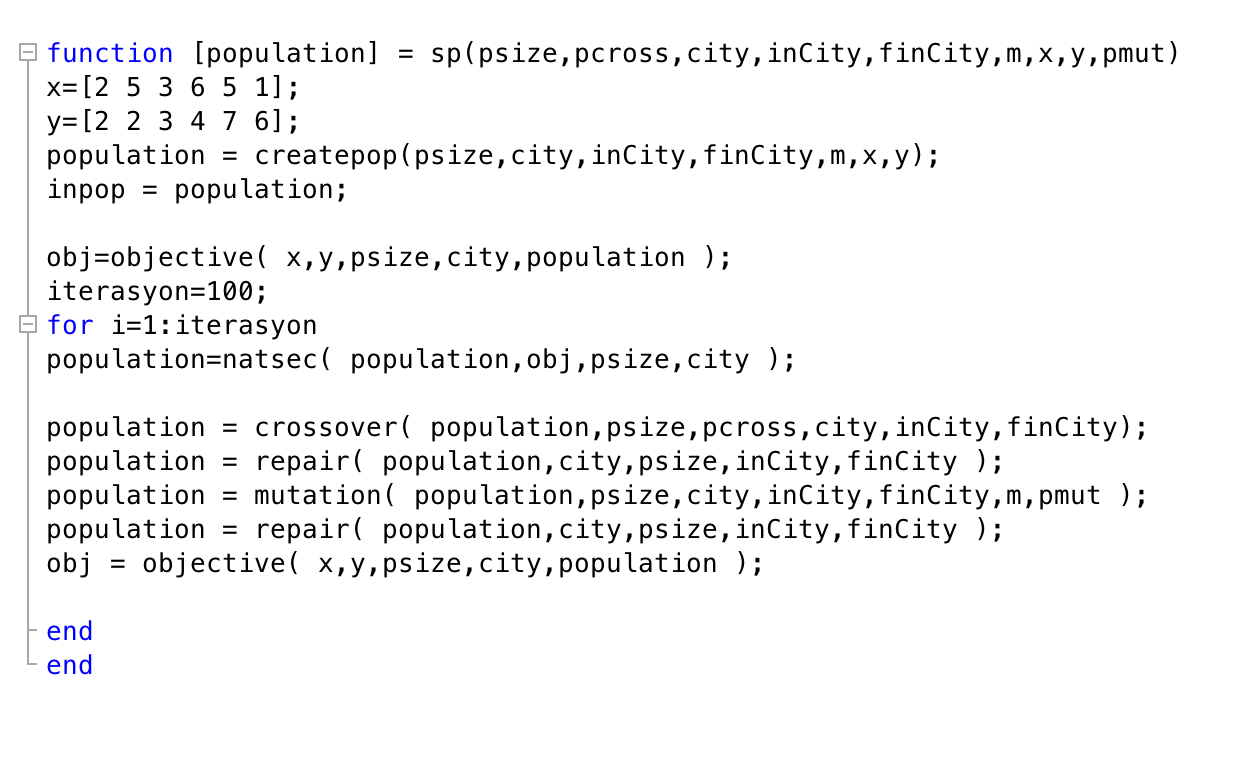


Figure 14. MATLAB code of the main function.

## **CONCLUSION**

To conclude, we have written a code for shortest path problem using genetic algorithm. We have designed a simple map with 6 cities. Inputs “city”,”psize”,“pcross”,”pmut”,“inCity”,”finCity” must be defined. We have observed that values of psize and iteration are quite important to obtain shortest path. For 6 cities, population size of 4 might be not enough to get shortest path but psize of 20 would be enough. Similarly, with few number of iteration output population might not contain shortest path.On the other hand, as iteration number increased, we observed quite reliable results.

We also used Dijkstra’s Algorithm to solve the same problem.The main difference is that Dijkstra’s Algorithm calculates all possible routes and chooses the shortest one whereas GA never calculates all possible routes. We may conclude that for the systems with low number of cities Dijkstra’s Algorithm might be more useful however it cannot be applied to systems with medium or high number of cities. The reason is because calculation of all possible routes will take quite amount of time. As a result we can say that for the systems with high number of cities GA is more useful with small amount of inaccuracy in return high amount of time saving.

## **APPENDIX**

**Population Creator Function**

***% DESCRIPTION:***

***% This function is our main function that contains several functions:***

***% Functions:***

***% createpop , objective , natsec , crossover , repair , mutation***

***%INPUTS:***

***% psize: Population size (must be an even number)***

***% pcross: probability of cross-over,***

***% must be between 0-1 (0.60-0.90 recommended) (0.5 means %50 chance)***

***% city: # of cities in the network (must be consistent with x and y***

***%matricies)***

***% inCity: Starting point***

***% finCity: Destination point***

***% m is the multiplied 1 and 0 matricies that indicates the connections***

***% between cities***

***% x is the matrix that holds the x axis coordinates of the cities***

***% y is the matrix that holds the y axis coordinates of the cities***

***% probability of mutation must be between***

***% must be between 0-1 (0.05-0.10 recommended) (0.5 means %50 chance)***

***% OUTPUT:***

***% popuation: A matrix that has psize number of solutions of our***

***% problem***

***function [population] = sp(psize,pcross,city,inCity,finCity,m,x,y,pmut)***

***x=[2 5 3 6 5 1];***

***y=[2 2 3 4 7 6];***

***population = createpop(psize,city,inCity,finCity,m,x,y );***

***inpop = population;***

***obj=objective( x,y,psize,city,population );***

***iterasyon=100;***

***for i=1:iterasyon***

***population=natsec( population,obj,psize,city );***

***population=crossover( population,psize,pcross,city,inCity,finCity);***

***population = repair( population,city,psize,inCity,finCity );***

***population = mutation( population,psize,city,inCity,finCity,m,pmut );***

***population = repair( population,city,psize,inCity,finCity );***

***obj=objective( x,y,psize,city,population );***

***end***

***end***

**Objective Function**

***% DESCRIPTION:***

***% This function is our objective calculator, in our case objective is the***

***% distance from starting point to the destination city.***

***% OUTPUT:***

***% obj is a psize x 1 matrix that holds the objective value of each cromosom***

***% in population.***

***function [ obj ] = objective( x,y,psize,city,population )***

***obj=zeros(psize,1);***

***for i=1:psize***

***for j=1:city***

***if(j)~=(city)***

***idx1=population(i,j); %bu ad?m di?er t?m yollar i?in.***

***idx2=population(i,j+1);***

***else***

***break***

***end***

***if idx2==0***

***break***

***end***

***obj(i)=sqrt((x(idx1)-x(idx2))^2+(y(idx1)-y(idx2))^2)+obj(i);***

***end***

***end***

***end***

**Natural Selection Function**

***% DESCRIPTION:***

***% natsec is Natural Selection function that takes population as input***

***% and creates another population(subpop) by using natural selection***

***% operation***

***% OUTPUT: subpop is our output variable. Its usage is the same as population***

***function [ subpop ] = natsec( population,obj,psize,city )***

***obj=1./obj;***

***sumobj=sum(obj);***

***probs=obj/sumobj;***

***cprobs=probs;***

***for i=2:psize***

***cprobs(i)=cprobs(i-1)+probs(i);***

***end***

***rs=unifrnd(0,1,[psize,1]);***

***subpop=population;***

***for i=1:psize***

***idx=find(rs(i)<cprobs,1);***

***subpop(i,:)=population(idx,:);***

***end***

***end***

**Crossover Function**

***%DESCRIPTION***

***% The crossover function generates 2 children from 2 parents.There will be***

***% crossover only if there is a common gene in both parents except starting***

***% and destination point genes.But there is no requirement that the genes***

***% need to be located at the same column.***

***% Example: P1: 2 4 5 6 0 0 P2: 2 3 4 6 0 0***

***% C1:2 4 6 0 0 0 C2: 2 3 4 5 6 0***

***function [subpop ] = crossover( population,psize,pcross,city,inCity,finCity)***

***pairs=randperm(psize);***

***for i=1:psize/2***

***idx1=pairs(2\*i-1);***

***idx2=pairs(2\*i);***

***parent1=population(idx1,:);***

***parent2=population(idx2,:);***

***rs=unifrnd(0,1);***

***children1=parent1;***

***children2=parent2;***

***if(rs<pcross)***

***ia=randperm(city\*2);***

***spar2=parent2(ia);***

***a=1;***

***while a<=2\*city***

***for b=2:2\*city***

***if ( parent1(b)==spar2(a) && spar2(a)~= 0 && spar2(a)~=inCity && spar2(a)~= finCity )***

***t=parent1(b);***

***loc=find(parent2==t,1);***

***children1=parent1;***

***children2=parent2;***

***c1=city-loc;***

***c2=city-b;***

***children1(b+1:b+c1)=parent2(loc+1:city);***

***children1(b+c1+1:end)=zeros;***

***children2(loc+1:loc+c2)=parent1(b+1:city);***

***children2(loc+c2+1:end)=zeros;***

***a=2\*city+1;***

***break***

***end***

***end***

***a=a+1;***

***end***

***subpop(idx1,:)=children1;***

***subpop(idx2,:)=children2;***

***else***

***children1=parent1;***

***children2=parent2;***

***subpop(idx1,:)=parent1;***

***subpop(idx2,:)=parent2;***

***end***

***end***

***end***

**Mutation Function**

***%DESCRIPTION***

***% The mutation function makes population change from a spesific point on.***

***% That point is obtained randomly and possibility of mutation depends pmut***

***% input value.(If population row cantains 2 cities, the possibility of mutation is zero.)***

***function [ population ] = mutation( population,psize,city,inCity,finCity,m,pmut )***

***ra=unifrnd(0,1);***

***if (ra<pmut)***

***for i=1:psize***

***if (population(i,2)==finCity)***

***continue***

***end***

***t=1;***

***while t==1***

***a=randi(city);***

***if a~=0 && a~=inCity && a~=finCity***

***loc=find(a==population(i,:));***

***l=length(loc);***

***if l~=0***

***mut=population(i,:);***

***mut(1:loc)=population(i,1:loc) ;***

***c=2\*city-loc+1;***

***newpop=createpop(1,city,a,finCity,m );***

***mut(loc:end)= newpop(1,1:c);***

***population(i,:)=mut(1,:);***

***t=t-1;***

***end***

***end***

***end***

***end***

***end***

***end***

**Repair Function**

***% DESCRIPTION:***

***% The main usage of this function is to repair infeasible chromosomes which***

***% are not satisfying our condition .***

***% Example:***

***% chromosome1=[4 2 1 5 6 2 3] as can be seen there is a loop in this***

***% solution which is not acceptable. The repaired choromosome should be***

***% something like that choromosome1(repaired)=[4 2 3]***

***function [ population] = repair( population,city,psize,inCity,finCity )***

***for i=1:psize***

***repsolution=population(i,:);***

***for j=2:2\*city***

***if population(i,j) == 0 || population(i,j)==finCity***

***break***

***else***

***num=population(i,j);***

***a=find(num==population(i,:));***

***l=0;***

***l=length(a);***

***if l>1***

***repsolution(1:a(1))=population(i,1:a(1));***

***k=2\*city-a(l)-1;***

***repsolution(a(1)+1:a(1)+1+k)=population(i,a(l)+1:end);***

***repsolution(a(1)+2+k:end)=zeros;***

***end***

***end***

***end***

***population(i,:)=repsolution;***

***end***

***end***

**Main Function**

***% DESCRIPTION:***

***% This function is our main function that contains several functions:***

***% Functions:***

***% createpop , objective , natsec , crossover , repair , mutation***

***%INPUTS:***

***% psize: Population size (must be an even number)***

***% pcross: probability of cross-over,***

***% must be between 0-1 (0.60-0.90 recommended) (0.5 means %50 chance)***

***% city: # of cities in the network (must be consistent with x and y***

***%matricies)***

***% inCity: Starting point***

***% finCity: Destination point***

***% m is the multiplied 1 and 0 matricies that indicates the connections***

***% between cities***

***% x is the matrix that holds the x axis coordinates of the cities***

***% y is the matrix that holds the y axis coordinates of the cities***

***% probability of mutation must be between***

***% must be between 0-1 (0.05-0.10 recommended) (0.5 means %50 chance)***

***% OUTPUT:***

***% popuation: A matrix that has psize number of solutions of our***

***% problem***

***function [population] = sp(psize,pcross,city,inCity,finCity,m,x,y,pmut)***

***x=[2 5 3 6 5 1];***

***y=[2 2 3 4 7 6];***

***population = createpop(psize,city,inCity,finCity,m,x,y );***

***inpop = population;***

***obj=objective( x,y,psize,city,population );***

***iterasyon=100;***

***for i=1:iterasyon***

***population=natsec( population,obj,psize,city );***

***population=crossover( population,psize,pcross,city,inCity,finCity);***

***population = repair( population,city,psize,inCity,finCity );***

***population = mutation( population,psize,city,inCity,finCity,m,pmut );***

***population = repair( population,city,psize,inCity,finCity );***

***obj=objective( x,y,psize,city,population );***

***end***

***end***