INFO6205_301

GENETIC ALGORITHMS TO SOLVE PIPELINE OPTIMIZATION PROBLEM

Group 301

Hao Cui, Miner Yang, Ruisi Gu

TABLE OF CONTENTS

1 Explanation on problem and concepts

1.1Problem background:

1.2 Details of concepts:

1.2.1 Genotype
1.2.2 Expression
1.2.3 Phenotype
1.2.4 Problem
1.2.5 Fitness

1.3 Core concepts

1.3.1 Encoding
1.3.2 Decoding

2 Implementation

2.1 Data generator

2.2 Solution

2.2.1 Selection
2.2.2 Crossover
2.2.3 Mutation

3 Results & CONCLUSIONS

3.1 Results

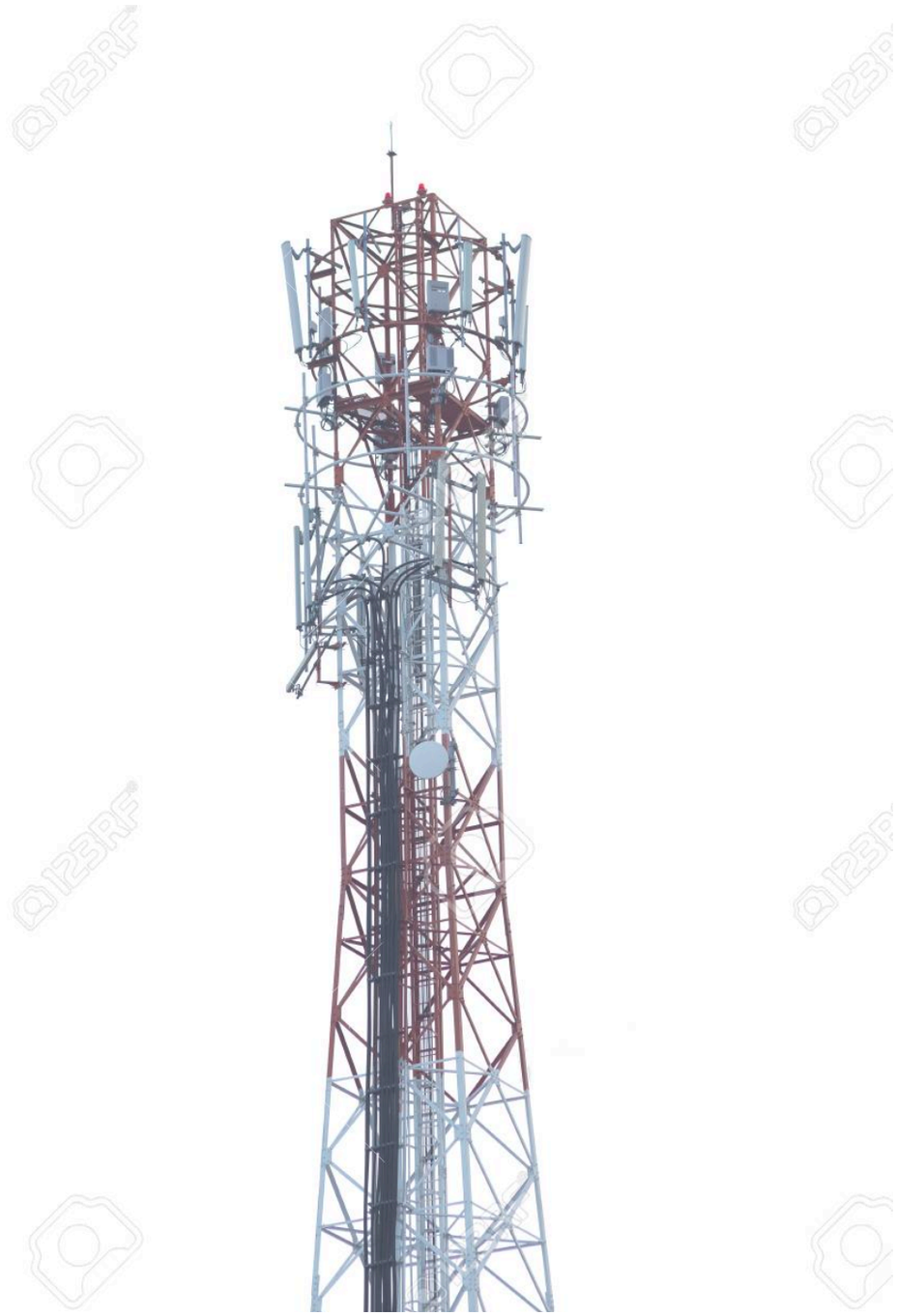
3.1.1 Unit Test
3.1.2 Output

3.2 Conclusions

3.2.1 Graphs
3.2.2 Conclusion 1
3.2.3 Conclusion 2

1 EXPLANATION ON PROBLEM AND CONCEPTS

- 1.1 Problem background
- 1.2 Details of concepts
- 1.3 Core concepts
  - 1.3.1 Encoding
  - 1.3.2 Decoding

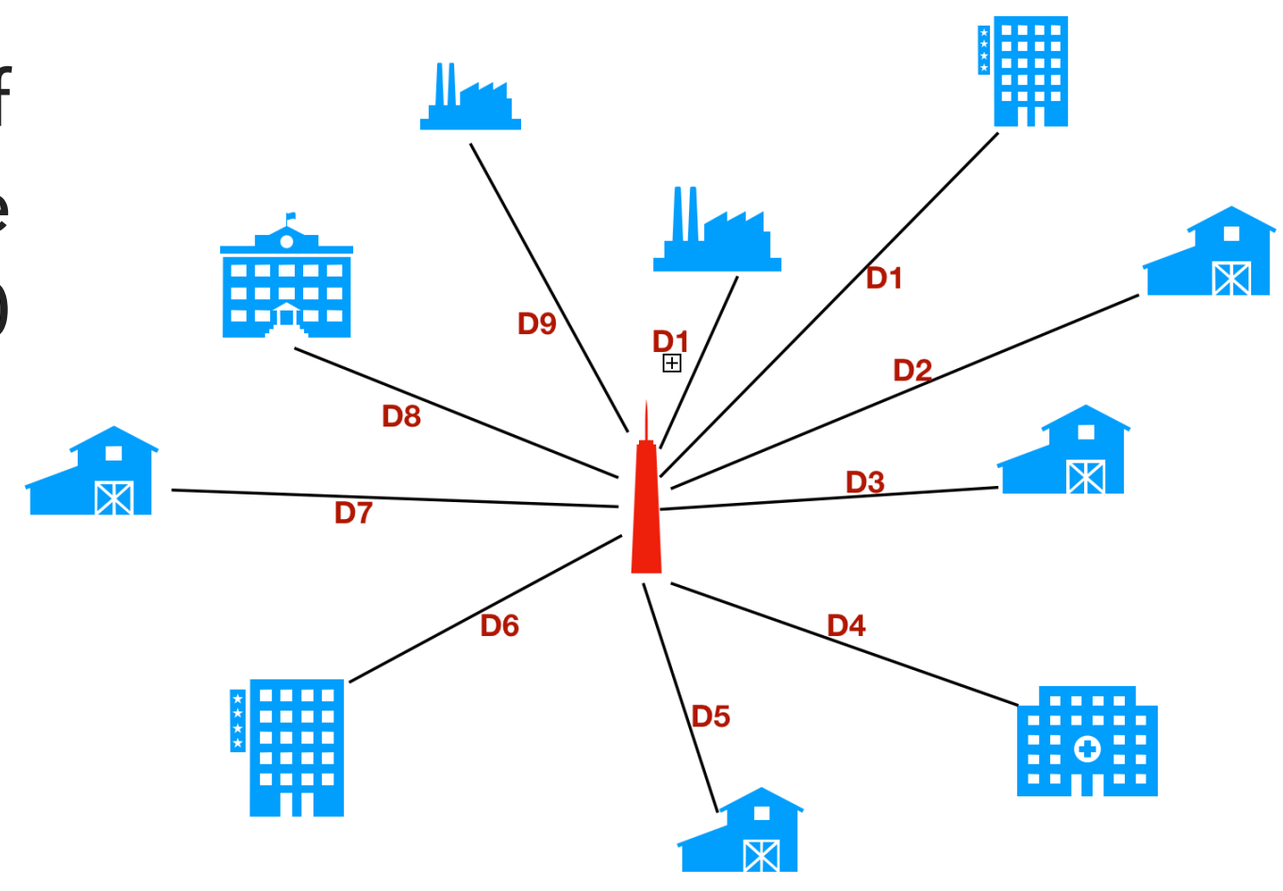


- In city B, there are ten known users of our grid company. Our job is to establish a base station for these users to satisfy their needs of network signal. However, considering the cost efficient and other factors, the relationship of infeasibility score and the sum of length from the base station to ten users is a parabola going upwards. The company will decided the location of base station according to the infeasibility score.

First, we need to calculate the sum of geometric distance from base station to ten known users, formulate as follow:

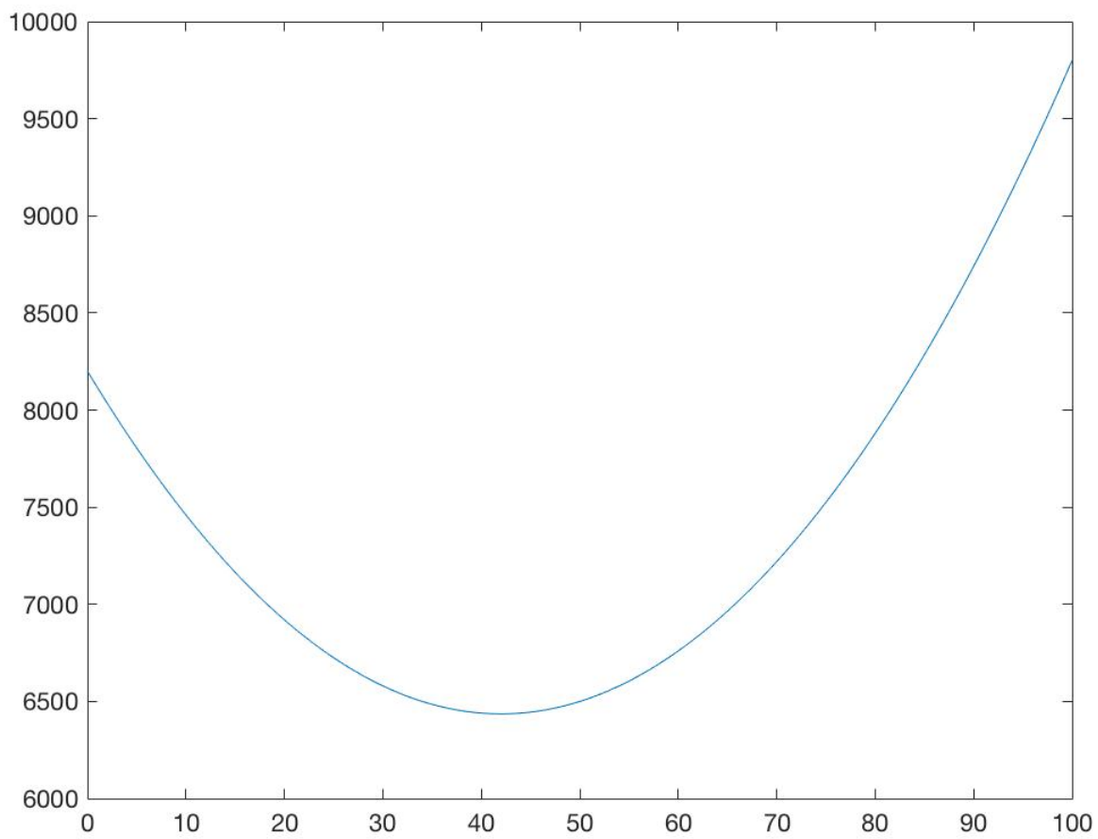
$$D(x_i, y_i) = \sum_{i=0}^{10} \sqrt{{x_i}^2 + {y_i}^2}$$

Where D denote sum of distance, x and y denote the coordinate of one of the 10 known points.



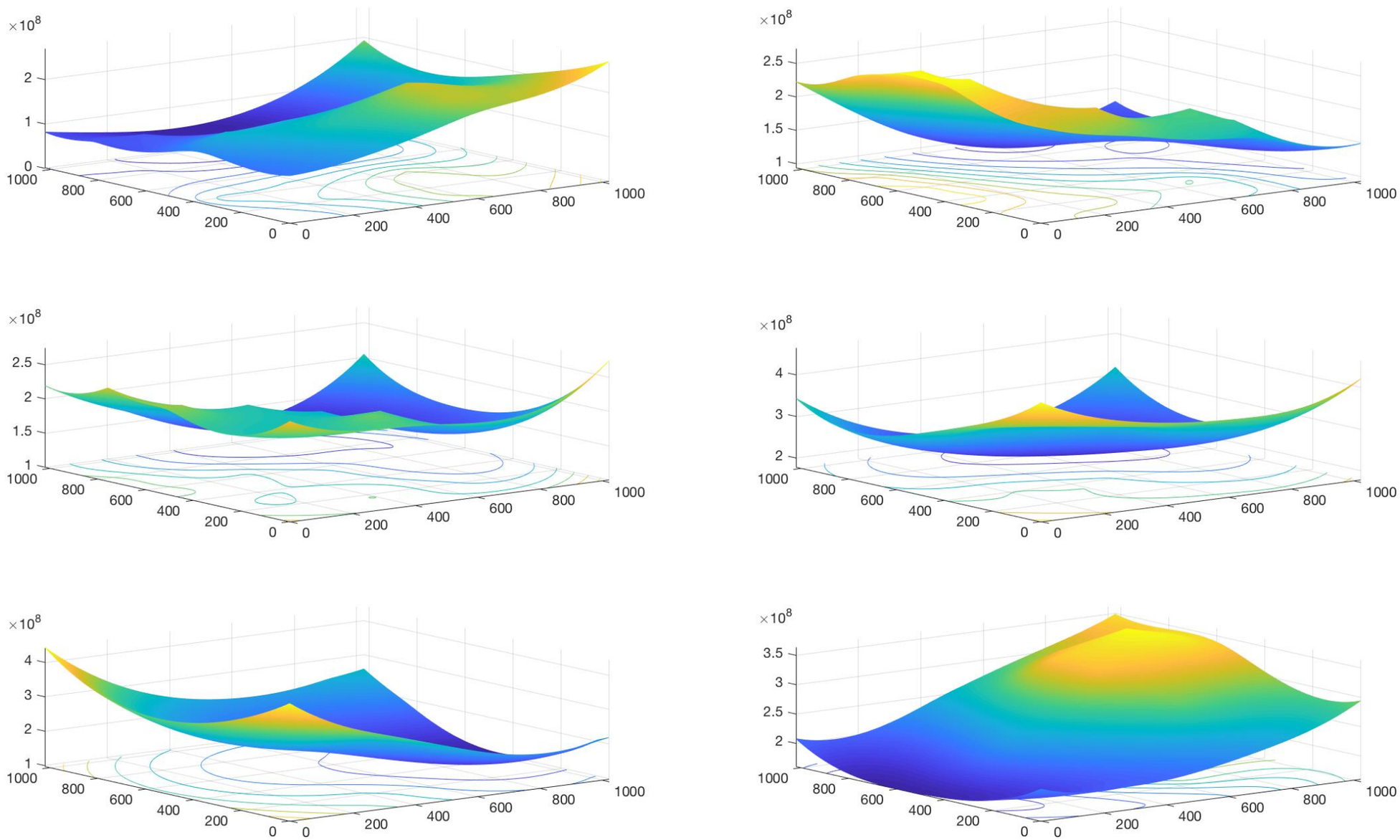
Relation between degree of infeasibility and the total distance of wire is quadratic:

$$S(D)=(D-C_1)^2+C_2*D+C_3$$



Where S denote degree of infeasibility, D denote distance from base station to users, C denote constant.

Thus, the relation between degree of infeasibility and the coordinate value will be shown as 3-dimension curved surface with multiple peaks.



0 1 0 1 1 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0

- 1.2.1 Genotype: Since the whole program is based on the coordinate of 1024*1024, the format of the genotype will be presented as binary numbers with 10 + 10 bits. We can separate this genotype into two parts, the first 10 digital will be the Y-axis, and the last 10 digital will be the X-axis.

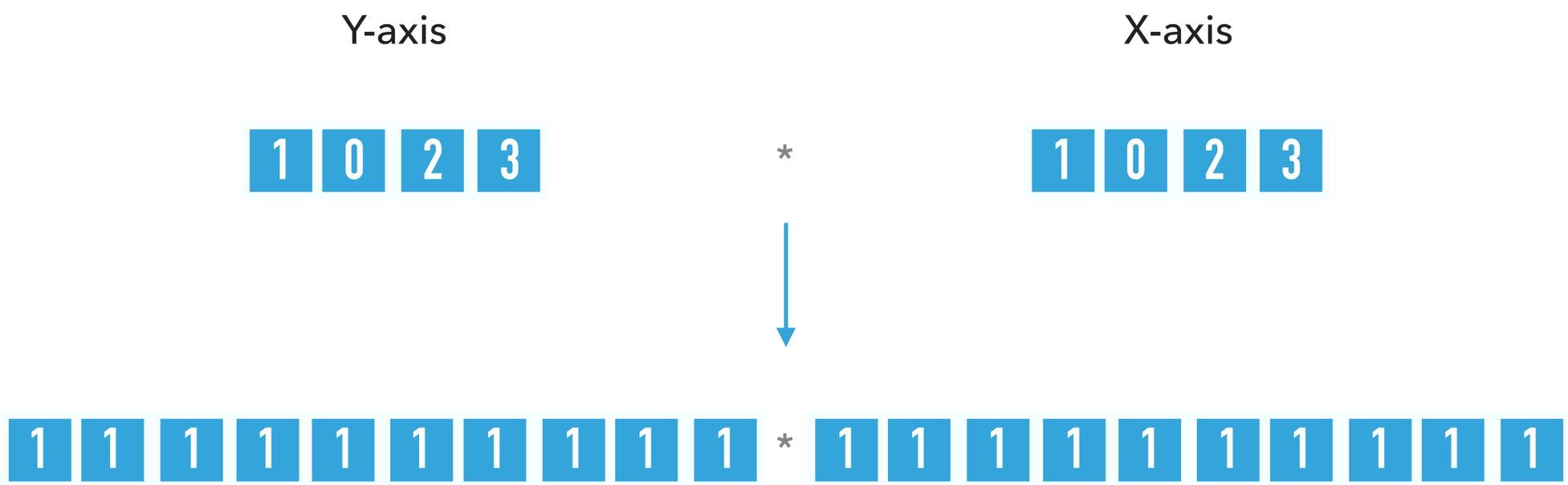
- 1.2.2 Expression: The expression will be presented in a three-dimensional coordinate system. The X-axis is the last 10 digital of the genotype of the certain point, the Y-axis is the first 10 digital of the genotype of the point, the Z-axis is the feasibility of the point.

- 1.2.3 Phenotype: The score of infeasibility.
- 1.2.4 Problem: We are trying to find out the point which has the lowest infeasibility score regarding several factors such as cost efficiency On the basis of ten known points.
- 1.2.5 Fitness: In our model, we are looking for the point which has the minimum score of infeasibility. The relationship between infeasibility score and the sum of length is quadratic equation instead of liner equation. Thus, the lower infeasibility score, the better fitness it has and the point has higher chosen probability, which means this points should holds higher ratio in the random process.

1.3.1 ENCODING

- ENCODING: We regard a point(X,Y) as an individual, and basically encode every coordinates into a 8-digit integer which first 4 digits represent value of Y, and last 4 digits represent Y. And we use an array to store the encipher coordinates. Encoding formula is shown as below:
  - Point(X,Y) –>integer
  - integer=Y*10000+X
- Example: P(229,360)->int A=360*10000+229 =3600229

EXPLANATION OF ENCODING



- Data range of encoded value 1024*1024=1million
- Value of each coordinate is 1024, which will transfer it into a binary digits within 10bits.

1.3.2 DECODING

- DECODING: When we need to use a coordinate of a points, we have to decode the number into two parts, which higher 4 digits and lower 4 digits should be separated as following:
  - Integer–>Points(X, Y)
  - X=integer%10000 (Integer Modulo Operation)
  - Y=integer/10000 (Integer Division Operation)
- Example: A=3600229 ->P(x, y)

  x=3600229%10000=229

  y=3600229/10000=360

2 IMPLEMENTATION

- 2.1 Data Generator
- 2.2 Solution:
  - 2.2.1 Selection
  - 2.2.2 Crossover
  - 2.2.3 Mutation

2.1 DATA GENERATOR

- In the map of 1024*1024 grid, we randomly select 10 points from the coordinates as our known invariant points. The problem will be to find out the point which has the minimum sum of distances from itself to these ten points.
- Also in this grid , we randomly generate 1000 integer as our initial group sample. Each integer contains the coordinate information of a point . The encoding and decoding method has be mentioned in Chapter 1.3

2.2.1 SELECTION

- After having randomly selected 1000 points a_1, a_2, ..., a_{1000}, we use FuncCalc(x) to calculate the weight of each points recorded as S_1, S_2, ..., S_{1000}, The method of computing weight is mentioned in chapter 1.1. In order to get the optimal solution, we need to calculate the genetic probability of each points, which should satisfy the following three conditions:

  1. The total of probability would still remain 1.
  2. The smaller the weight S_i , the higher the probability of this point it will hold.
  3. Every point has it own probability.

2.2.1 SELECTION

- The calculation process is as follow:

- Calculate the rate *rat1* by using S_i divided by the sum of S. Currently, the larger the weight S_i, the larger the probability:

$$rat1[i] = \frac{S_i}{\sum_{i=1}^{1000} s_i}$$

- Get the rate *rat2* by countdown the *sumrat*. Now, the greater the weight, the smaller the probability. The weight and probability satisfy the inverse relationship:

$$rat2[i] = \frac{1}{rat1[i]}$$

- After having the *sumrat2*, we have to make sure that the total of *sumrat2* would still remain 1:

$$rate[i] = \frac{rat2[i]}{\sum_{i}^{1000} rat2[i]}$$

2.2.1 SELECTION

- According to the rate we have calculated, now reflect these probabilities into a subinterval of 0-25000, the definition of range is as follow:

$$range(left) = the\ range(right) of\ the\ prvious\ subinterval + 1$$
$$range(right) = rate[i] * 2{,}5000 + range(left)$$

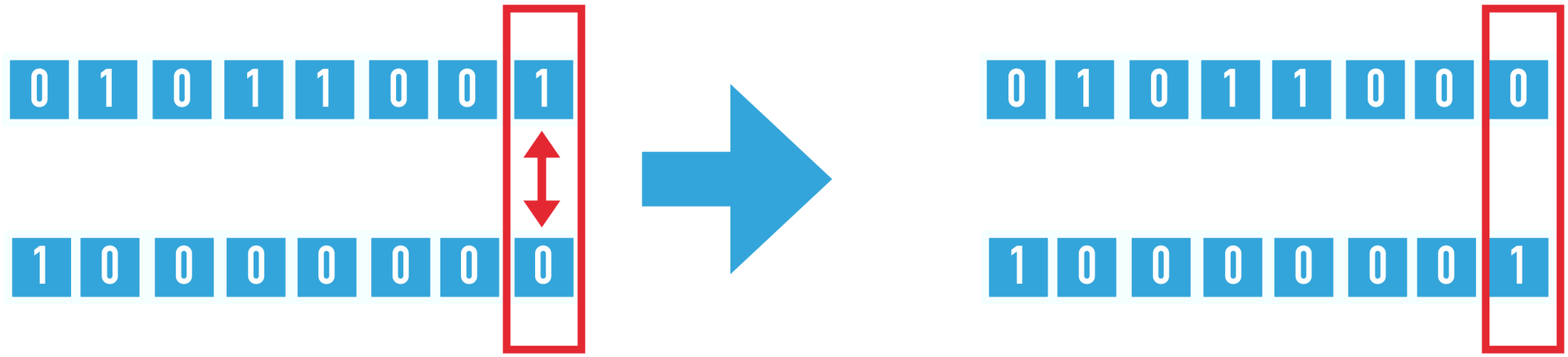
- The reason of number 2,5000:

- Lower limit: For 1000 units, there must be one unit whose probability is less than $\frac{1}{1000}$, so that the interval has to be larger than 1000
- Upper limit: Considering the time complexity of this algorithms, the number cannot be positive infinity. $O(n) = 2{,}5000 * 10{,}000$

2.2.2 CROSSOVER

- Before doing the crossover process, we need shuffle the 1000 solutions. So that the two solutions i and i+1 we pick out later to crossover will be randomly selected.
- First, change genotype into binary representation, therefore, original coordinate 1024*1024 then be converted into 20 bits. The first 10 bits represent the Y-axis, the last 10 bits represent the X-axis.
- Then, let *k* has two values 0 and 1. Randomly choose a value, if k equals 0, we do the crossover process on the first 10 bits, that is, Y-axis. When k equals 1, we do it on the last 10 bits, that is, X-axis.
- After having decided the range of target, we randomly choose a location from 10 bits, recorded as *tempx*. Exchange the *tempx* bit of i and i+1 as the crossover process.

2.2.2 CROSSOVER



>> Randomly choose two individual to A and B to crossover

>>Randomly generate the target-mutated bit M from 0 to 9

>>Randomly choose to exchange the coordinate in X-axis or Y-axis

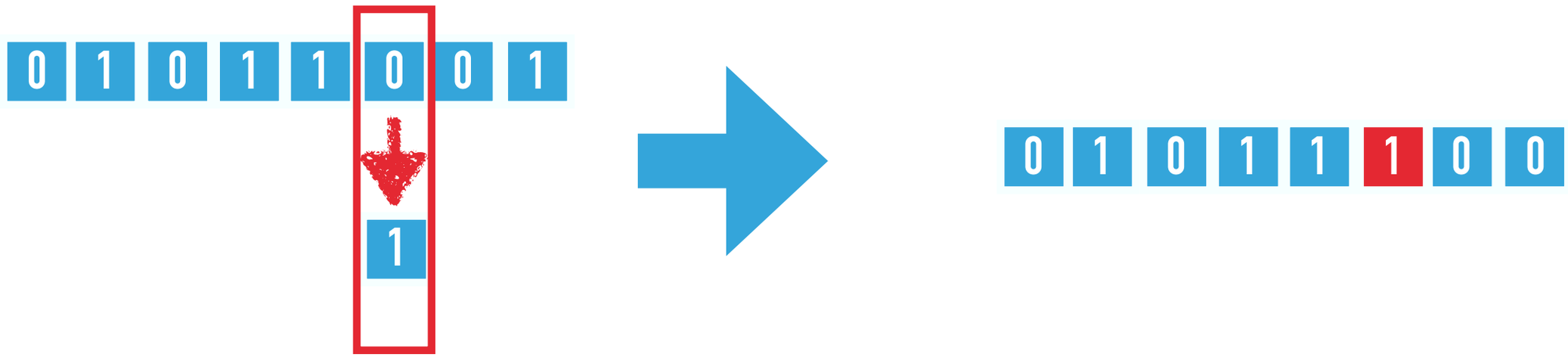
>>Exchange the Mth bit of A and B, which is shown above

Crossover Example

2.2.3 MUTATION

- First, select the position to mutate from 20 bits recorded as *temloc*. This process also randomly decide either X-axis or Y-axis to exchange.
- Then let the gene mutation rate be 2% by selecting 1 from 50 integer. When mutation happening, if the value on *tmploc* equals 0, then change this bit into 1, else if the value equals 1 change it into 0.

2.2.3 MUTATION



>> Randomly generate a integer from 0-19 as the target-mutated bit M

>> if 0<M<9, we change the Mth bit of x coordinate

>> if 10<M<19, we change the (M-10)th bit of y coordinate

>> As for change, we transfer 1 to 0, 0 to1

Mutation Example

3 RESULTS & CONCLUSIONS

- 3.1 RESULTS
  - 3.1.1 Unit Test
  - 3.1.2 Output
- 3.2 CONCLUSIONS
  - 3.2.1 Graph
  - 3.2.2 Conclusion1
  - 3.2.3 Conclusion2

3.1.1 UNIT TEST

1

Tests passed: 8 of 8 tests – 1 m 33 s 662 ms

GAtest (pkg6205	1 m 33 s 662 ms
CrossOverCheck	111 ms
OptimalSolution	32 s 470 ms
MutationCheck	45 ms
DataRangeCheck	124 ms
RatioBoundCheck	96 ms
MinimumCheck	30 s 270 ms
EvolveBetter	30 s 456 ms
SelectCheck	90 ms

```
/Library/Java/JavaVirtualMachines/jdk-10.0.2.jdk/Contents/Home/
 [0]129
 [1]88
actual min9.723520988350967E7
goal min9.723068459622152E7
1.360673062513917E8 1.3635650475706947E8 1.3640797519752762E8
1.36064152052987E8 1.3606944223679972E8 1.3606944223679972E8
avg1 2.830499534650716E8
avg2 1.8287236493965143E8

Process finished with exit code 0
```

2

Tests passed: 8 of 8 tests – 1 m 36 s 219 ms

GAtest (pkg6205	1 m 36 s 219 ms
CrossOverCheck	148 ms
OptimalSolution	35 s 143 ms
MutationCheck	41 ms
DataRangeCheck	101 ms
RatioBoundCheck	77 ms
MinimumCheck	30 s 726 ms
EvolveBetter	29 s 918 ms
SelectCheck	65 ms

```
/Library/Java/JavaVirtualMachines/jdk-10.0.2.jdk/Contents/Home/b
 [0]129
 [1]88
actual min1.373166016181647E8
goal min1.372880487346506E8
1.1974383943384229E8 1.197941828023488E8 1.2006746064226325E8
1.1974383943384229E8 1.1974487050586993E8 1.1974490867809686E8
avg1 1.8723401375251952E8
avg2 5.183517357814863E7

Process finished with exit code 0
```

3

Tests passed: 8 of 8 tests – 1 m 32 s 958 ms

GAtest (pkg6205	1 m 32 s 958 ms
CrossOverCheck	221 ms
OptimalSolution	31 s 75 ms
MutationCheck	36 ms
DataRangeCheck	88 ms
RatioBoundCheck	74 ms
MinimumCheck	31 s 110 ms
EvolveBetter	30 s 276 ms
SelectCheck	78 ms

```
/Library/Java/JavaVirtualMachines/jdk-10.0.2.jdk/Contents/Hom
 [0]129
 [1]88
actual min1.0041893968794054E8
goal min1.0040448432187551E8
8.698563285597654E7 8.708403492896417E7 8.710435628036778E7
8.6921692723819E7 8.6921692723819E7 8.6932939115639E7
avg1 1.9431423642756587E8
avg2 1.445933074907535E8

Process finished with exit code 0
```

3.1.1 UNIT TESTS

- We conduct 8 unit tests including 4 core function tests and 4 performance tests. You can refer to the attached code file GAtest.java.
- <u>FOUR core function test</u>
  - Aim to check the validity of data , individual chosen-probability and other critical part of processes such as mutation and cross over.
  - Function name:
    - DataRangeCheck()
    - RatioBoundCheck()
    - CrossOverCheck()
    - MutationCheck()

3.1 RESULTS

- FOUR performance test
  - **SelectCheck():** To check if less weight one individual holds, the higher chosen-probability it has.
  - **EvolveBetter():** Compare the average weight of original group and evolved group we finally got. The latter should be less.
  - **MinimumCheck():** To whether the minimum solution in the evolved group is less than the one in the original group
  - **OptimalSolution():**
    - a.Calculate the weight of 1024*1024(1 million) points, and find the minimum value as the ideal value A.
    - b.Find the minimum value in the evolved group as actual value.
    - c.Check the deviation of ideal value and actual value B, if devotion is less than 0.002, then we take it as good performance.
    - d.Deviation computing method is d=abs.(sqrt(A)-sqrt(B))/sqrt(A).
    - Since relation between our ratio and weight is quadratic. It is more precise to using the root of the value as comparing object.

3.1.2 OUTPUT

```
1.9798922385259846E8 1.9785139789790708E8 1.977149
1.9823185365423325E8 1.9809343358628497E8 1.979564
1.9847548039743036E8 1.983364659954684E8 1.9819884
1.9872010588599974E8 1.985804969321023E8 1.9844227
1.9896573192301863E8 1.9882552820210153E8 1.986867
1.9921236031082448E8 1.9907156161064363E8 1.989321
1.9945999285100707E8 1.9931859896216124E8 1.991785
1.9970863134440133E8 1.995666420603344E8 1.9942604
1.9995827759107915E8 1.998156927080828E8 1.9967449
2.0020893339034212E8 2.0006575270755783E8 1.999239
2.0046060054071328E8 2.0031682386013493E8 2.001744
2.0071328083992934E8 2.005689079664053E8 2.0042592
2.0096697608493263E8 2.0082200682616776E8 2.006784
2.0122168807186276E8 2.0107612223842084E8 2.009319
2.0147741859604862E8 2.0133125600135407E8 2.011864
2.01734169452E8 2.0158740991234E8 2.01442034454879
2.0199194243339905E8 2.0184458576792526E8 2.016986
2.0225073933309194E8 2.0210278536382237E8 2.019562
2.0251056194308016E8 2.02362010494901E8 2.02214841
2.0277141205451208E8 2.0262226295517924E8 2.024744
2.0303329145767397E8 2.0288354453781456E8 2.027351
2.032962019419813E8 2.031458570350953E8 2.02996892
2.0356014529596967E8 2.0340920223843142E8 2.032596
2.0382512330728638E8 2.0367358193834567E8 2.035234
2.0409113776268074E8 2.039389979244643E8 2.0378823
2.0435819044799522E8 2.042054519855079E8 2.0405409
2.0462628314815626E8 2.0447294590928194E8 2.043209
2.048954176471651E8 2.0474148148266795E8 2.0458892
2.051655957280882E8 2.0501106049161372E8 2.0485790
2.0543681917304823E8 2.0528168472112373E8 2.051279
2.0570908976321396E8 2.055533559552498E8 2.0539899
2.059824092787917E8 2.058260759770816E8 2.05671118
2.0625677949901456E8 2.0609984656873658E8 2.059442
2.0653220220213377E8 2.0637466951135057E8 2.062185
2.0680867916540825E8 2.0665054658506775E8 2.064937
2.0708621216509533E8 2.06927479569031E8 2.06770120
2.073648029764406E8 2.0720547024137184E8 2.0704751
2.0764445337366816E8 2.0748452037920034E8 2.073259
2.0792516512997064E8 2.0776463175859532E8 2.076054
```

```java
*/
public class Solution {

    int[] answer_list = new int[1000];
    int[] loc_list = new int[10];
    double[] rat = new double[1000];
    int[] bel = new int[26000];
    int alen = 1000;
    int [] f= new int[10];
    public Solution(int [] a){
        for (int i = 0; i < 10; i++)
            loc_list[i] = a[i];
    }

    public void init_answer(){

        int[] a = new int [1<<20];
        for (int i = 0 ; i<1024 ; i ++)
            for (int j = 0; j<1024; j++)
                a[i*1024+j] = i*10000+j;

        Random x = new Random();

        for (int i = 0 ; i < 10; i++)
            f[i] = x.nextInt(1024);

        for (int i = (1<<20) - 1 ; i>0; i --){
            int tmp = x.nextInt(i+1);
            int tmp2 = a[i];
            a[i] = a[tmp];
            a[tmp] =tmp2;
        }
        for (int i = 0 ; i<alen ; i ++){
            answer_list[i] = a[i];
            //System.out.println(FuncCalc(a[i]
        }
      /* for (int i = 0 ; i < 1024; i++)
```

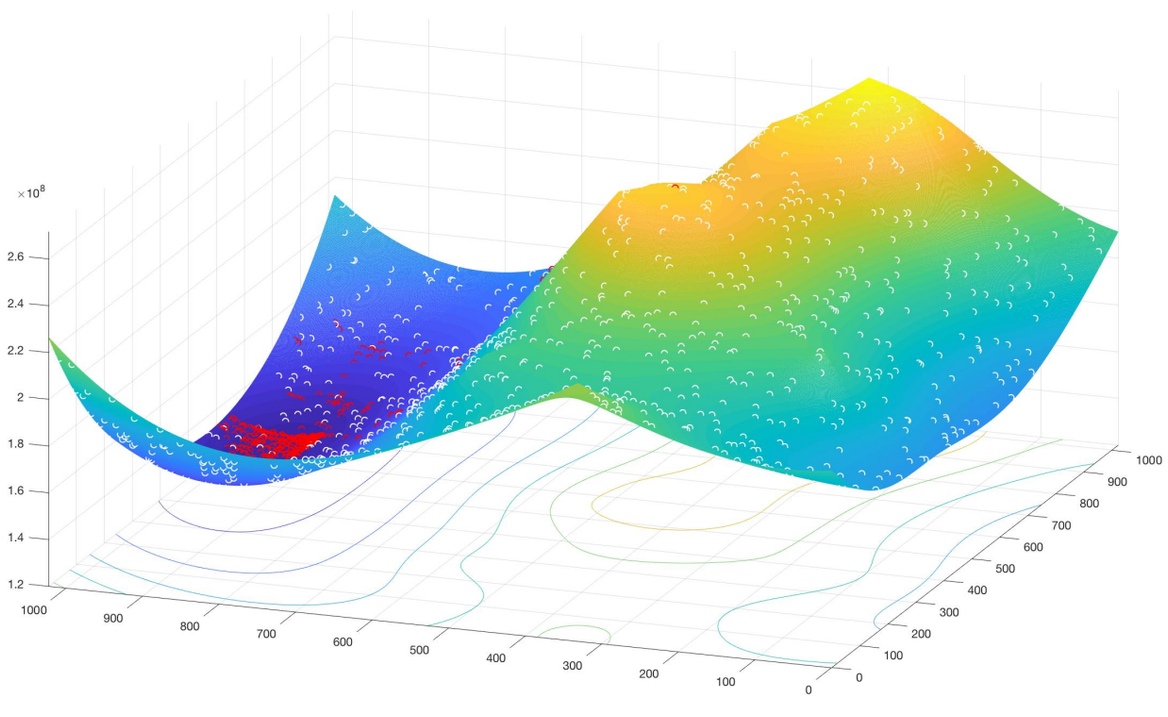
3.2.1 GRAPH

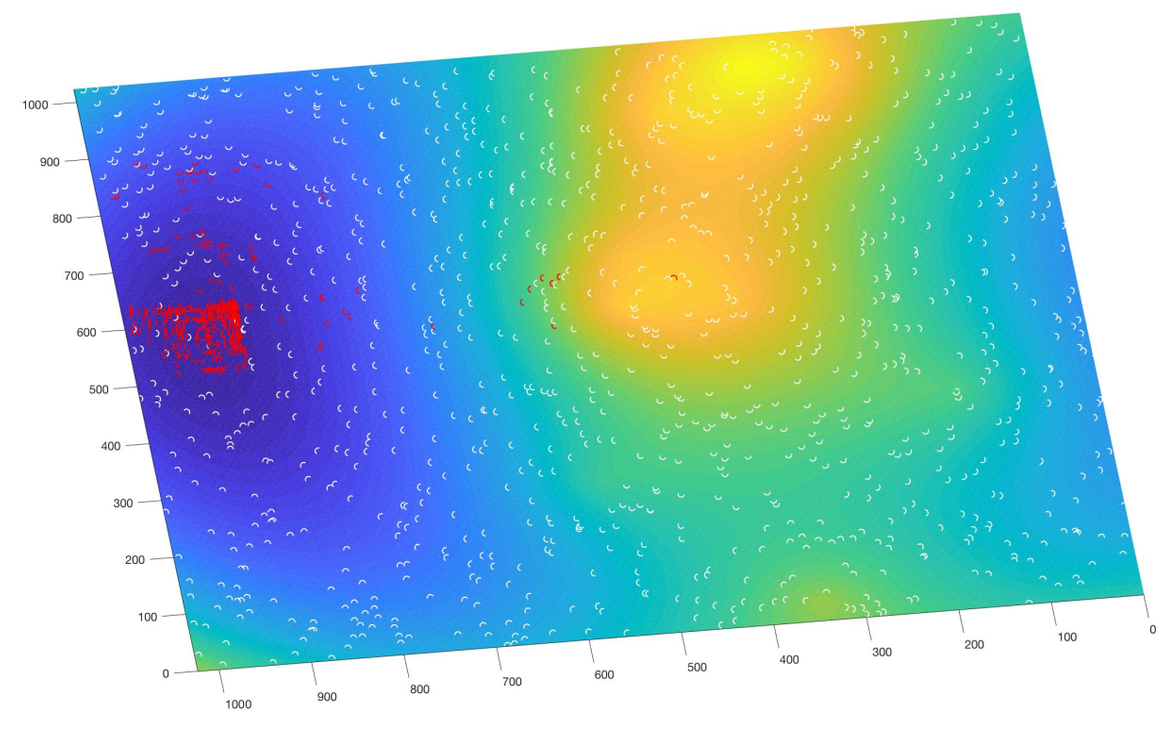
- As we can see from the graph, the solution has multiple peaks. The smaller Z-axis value, the lower infeasibility scores, the darker as shown on the graph, the better to select as our optimal points.
- Graphical Representation
  - X-axis: the last four digital of genotype before convert into binary numbers.
  - Y-axis: the first four digital of genotype before convert into binary numbers.
  - Z-axis: the score of infeasibility.
  - Colored points: 1000 ethnic groups evolved according to genetic algorithms.

3.2.2 CONCLUSION 1

- As we can see from the white points on the graph, 1000 solution is randomly diverge on the coordinate before the genetic algorithms process.

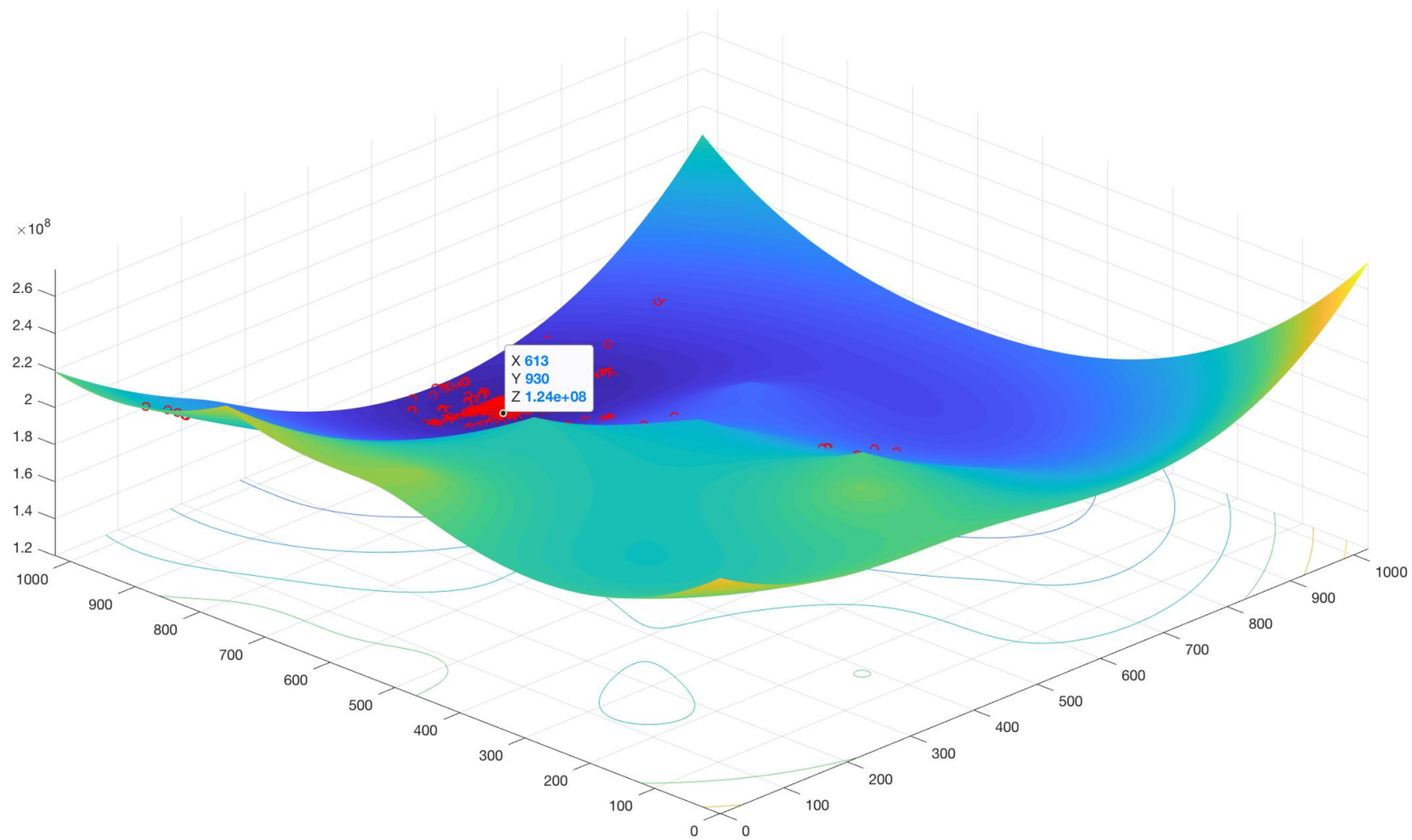
- After 100,000 times selection, crossover and mutation, the 1000 final results, which are red points now, convergence in one limited range .





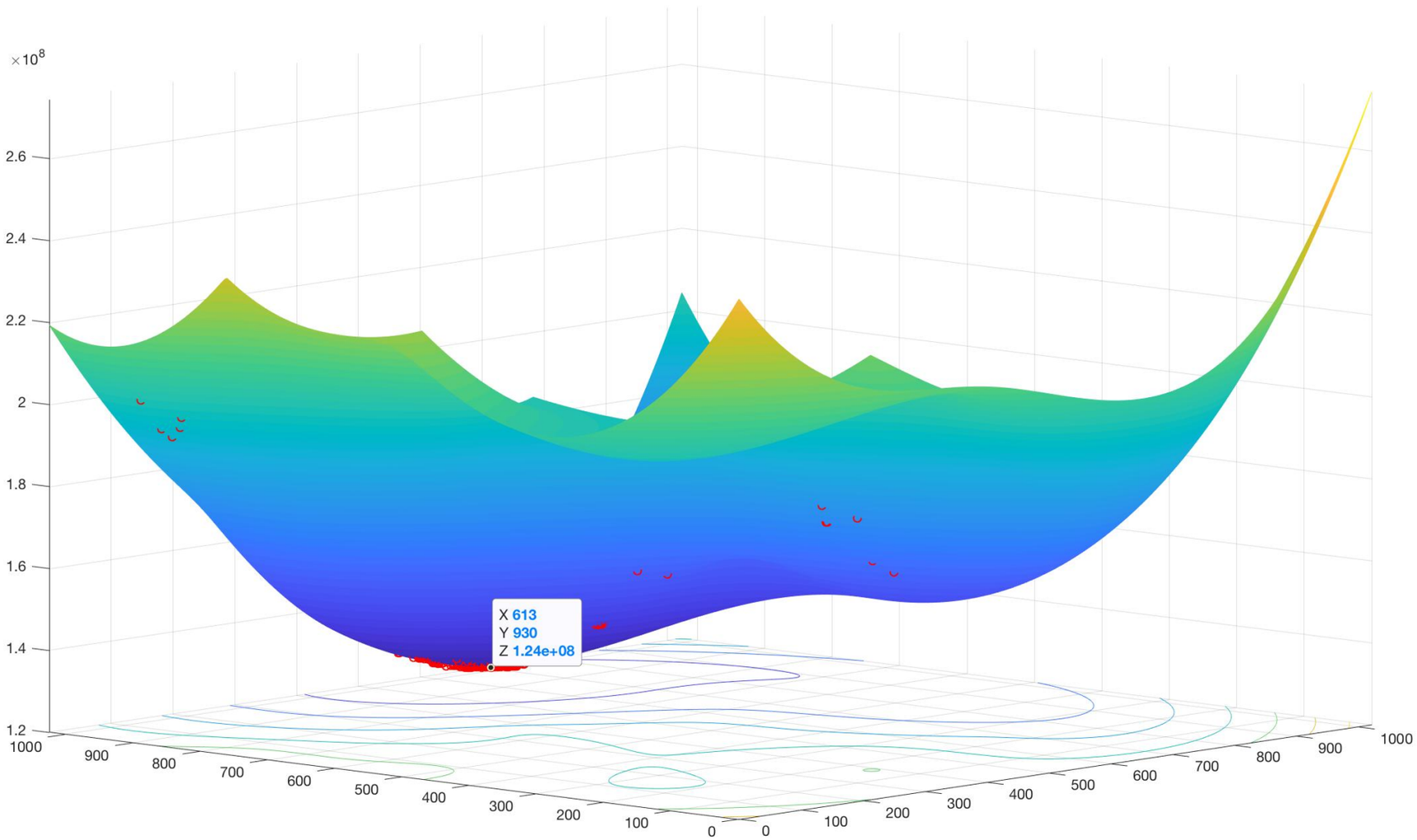
3.2.3 CONCLUSION 2

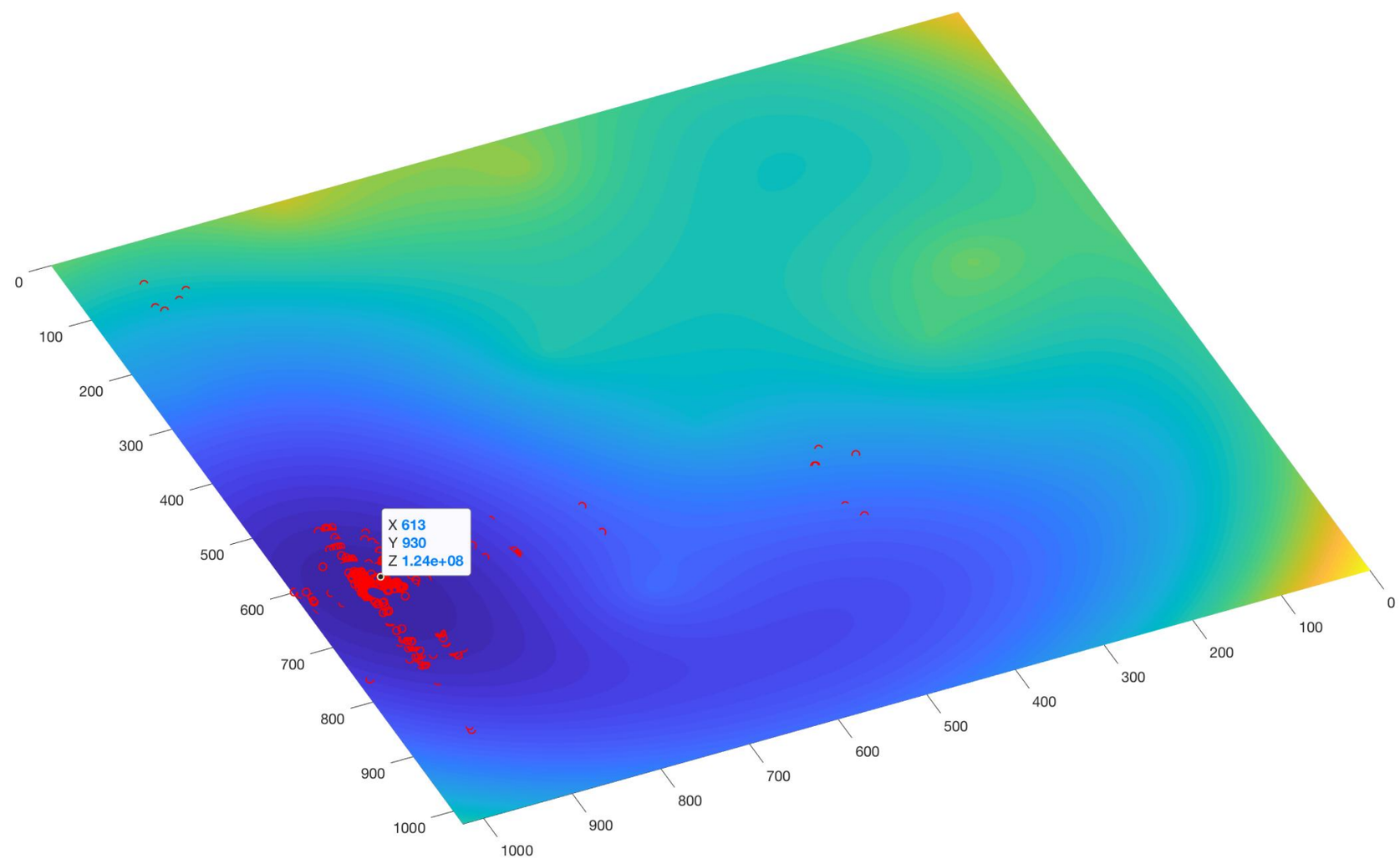
- These graphs represent the solution of selecting point after 100,000 times options from three different angles.
- The curved surface is made of the infeasibility score of 1 million points.



3.2.3 CONCLUSION 2

- The graph demonstrate that we have two lower peaks in the whole solutions group. Which means, the ideal point we are looking for should land up on one of these lower peaks.



3.2.3 CONCLUSION 2

- As we can see from the graphs, apart from some red points scattered elsewhere in the map, most of the red points (that is solutions calculated by genetic algorithms) are gathered in the lowest peak. Which means, our genetic algorithms generally perform well.