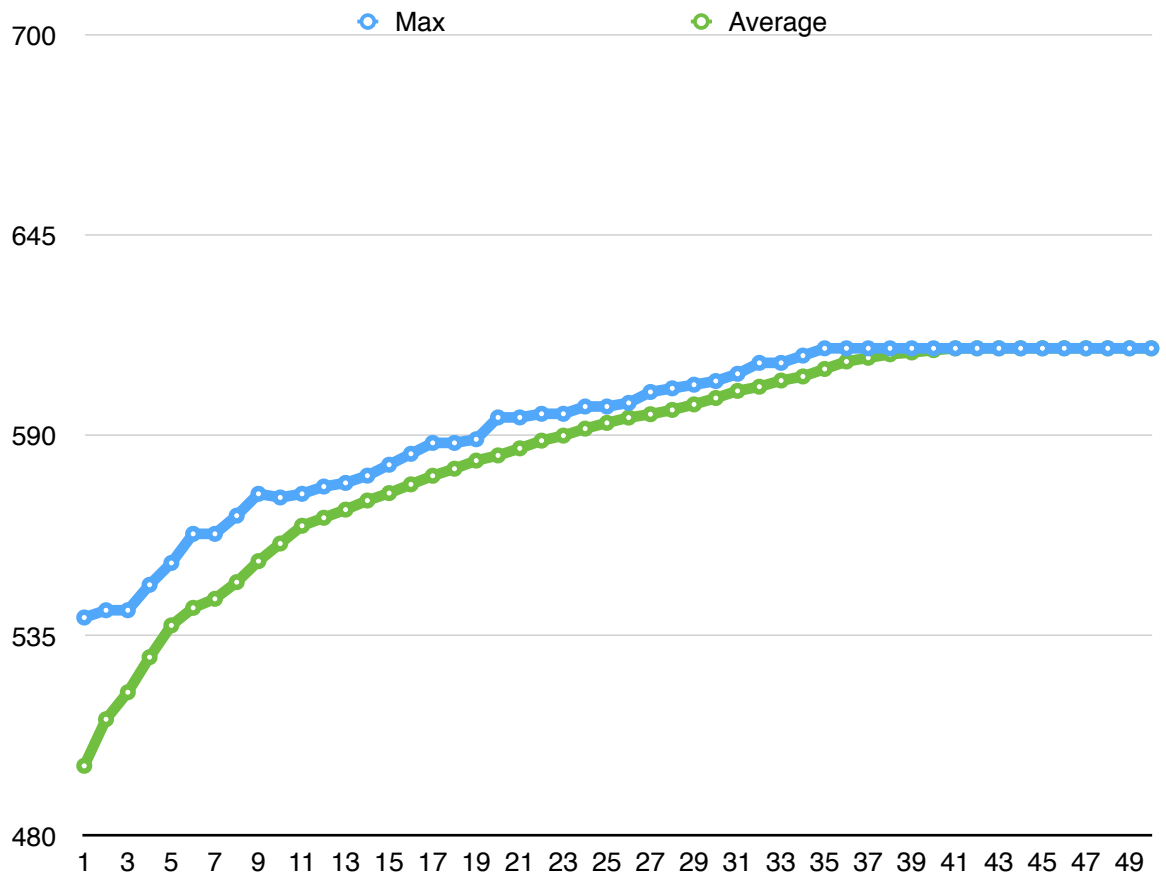
Graph for greatest chromosomes:



We can see at graph above, that after 50 iterations algorithm create chromosomes with 61.4% good gens. Average value equals with maximum value in the last part of iterations . When we run program again, we got a another results, its depends on function that generates initial chromosomes.

source code (JavaScript):

```
'use strict';


const genesCount = 1000;

const personCount = 100;
```

```
const sliceCount = 50;

const iterationCount = 50;


const averageFitness = [];

const maxFitness = [];


const rand = () => {

  const res = Math.random();

  return res > 0.5 ? 1 : 0;

};


const getRandomInt = (min, max) => {

  return Math.floor(Math.random() * (max - min + 1)) + min;

}


const generatePopulation = (count) => {

  return Array.from({ length: count }, v =>
generatePerson(genesCount));

};


const generatePerson = (count) => {

  return Array.from({ length: count }, v => rand());

};


const countFitness = (array) => {

  return array.reduce((p, c) => c == 1 ? ++p : p, 0);

};
```

```
const rotatePersons = (population) => {
  return population.sort((a, b) => countFitness(b) -
countFitness(a));
}

const getStatistics = (population) => {
  maxFitness.push(population.map(e => countFitness(e)).shift());
  averageFitness.push(getAverage(population));
}

const getAverage = (population) => {
  const all = population.reduce((p, c) => p += countFitness(c),
0);
  return all / personCount;
}

// reduce less fitness persons
const trimPersons = (population) => {
  return population.slice(0, sliceCount);
};

const getParents = (population) => {
  const firstParentindex = getRandomInt(0, sliceCount - 1);
  const secondParentindex = getRandomInt(0, sliceCount - 1);

  return {
    first: population[firstParentindex],
    second: population[secondParentindex],
  };
```

```
};


const crossover = (parents) => {
  const children = [parents.first, parents.second];
  const delimiter = getRandomInt(1, genesCount);


  children.push(parents.first.slice(0,
delimiter).concat(parents.second.slice(delimiter)));
  children.push(parents.second.slice(0,
delimiter).concat(parents.first.slice(delimiter)));


  return children;
};


const main = () => {
  let population = generatePopulation(personCount); // init


  for (let i = 0; i < iterationCount; i++) {
    population = rotatePersons(population);


    getStatistics(population);


    population = trimPersons(population);


    let parents;
    let newPopulation = [];


    for (let j = 0; j < sliceCount / 2; j++) {
      parents = getParents(population);
```

```
      const children = crossover(parents);
      newPopulation = newPopulation.concat(children);
    }

    population = newPopulation;
  }

  console.log('average', averageFitness);
  console.log('max', maxFitness);
}

main();
```

Table with results (iteration, average value of good genes, maximum value of good genes)

Iteration	average	maximum	
1	499.24	540	
2	512.04	542	
3	519.46	542	
4	529.1	549	
5	537.84	555	
6	542.66	563	
7	545.14	563	
8	549.72	568	
9	555.5	574	
10	560.34	573	
11	565.22	574	
12	567.42	576	

13	569.66	577	
14	572.16	579	
15	574.2	582	
16	576.6	585	
17	578.96	588	
18	580.9	588	
19	583.14	589	
20	584.56	595	
21	586.5	595	
22	588.62	596	
23	590	596	
24	591.92	598	
25	593.5	598	
26	594.96	599	
27	595.88	602	
28	597.06	603	
29	598.58	604	
30	600.34	605	
31	602.3	607	
32	603.44	610	
33	605.14	610	
34	606.26	612	
35	608.32	614	
36	610.38	614	
37	611.4	614	
38	612.36	614	
39	612.86	614	
40	613.48	614	
41	613.98	614	
42	614	614	
43	614	614	

44	614	614	
45	614	614	
46	614	614	
47	614	614	
48	614	614	
49	614	614	
50	614	614	