Parameter tuning of a genetic algorithm for finding central vertices in graphs

A Vlasov, A Khomchenko, A Faizliev, S Mironov and A Grigoriev
Saratov State University, Saratov, Russia
E-mail: mironovsv@sgu.ru

Abstract. This paper studies a genetic algorithm for finding the central vertices in graphs. The algorithm uses a different approach to the solution method presentation and describes a new insight in the crossover process. Studies are conducted to find the optimal parameters of the genetic algorithm such as crossover probability, mutation probability and population size. Based on the results, it can be claimed that with the right parameters, our algorithm shows good running time results with high accuracy of the correct answers.

1. Introduction

In recent times, with the introduction of computer technology and the emergence of new methods of data collection, the analysis of big data is becoming more acceptable. These studies reveal previously unexplored features of the communication of large groups of interacting objects. These data can represent not only theoretical, but also colossal practical benefits. It is obvious that for this kind of research it would be logical to have an appropriate algorithm that could work with big volumes of data in an acceptable amount of time. Varieties of big data include large graphs that describe social networks or the relationships between a large number of people.

In graph theory, finding the graph radius and central vertices is one of the important issues in theoretical and applied tasks.

This work considers an unweighted undirected graph \( G = (V, E) \), where \( |V| = n \) is the number of vertices, \( |E| = m \) is the number of edges. Since the graph is unweighted, the path length between the vertices \( u \) and \( v \) is equal to the number of edges in it.

To find the central vertex in the graph, we can consider the concept of vertex eccentricity, i.e. the distance from the vertex to the most distant vertex. Then the graph radius is the minimum of the eccentricities of all the vertices. Besides, the vertex on which this minimum is reached is called central vertex.

This paper considers connected graphs, since if a graph is not connected, then there is a pair of vertices with infinite distance, and in this case the task does not make sense.

This problem is well examined, and there are several exact algorithms that are used to search for the central vertices of the graph. In addition, together with the problem of searching for central vertices in a graph, the problem of all-pairs shortest path in a graph is often considered.

On the one hand, this task may seem trivial. There is a breadth-first search (BFS) algorithm that allows one to find the distance from one vertex to all other vertices with a time estimate \( O(n+m) \). If the algorithm runs from each vertex, it can solve the problem and obtain the exact
solution. At the same time, it is easy to notice that the time estimate of the trivial algorithm is $O(nm)$, which at high graph density is $O(n^3)$.

At its core, all other works were written to improve this time estimate. There are algorithms that allow one to solve this problem in less time. In general, there are two main approaches that were proposed for finding central vertices in graphs.

The first one is proposed in [1] and is based on matrix multiplication. In this algorithm, the graph is represented as an adjacency matrix. The algorithm is based on working with a matrix, and the most resource-consuming operation in its transformations is matrix multiplication. Accordingly, a time estimate of this algorithm is $O(n^3)$, but if combined with fast multiplication algorithm, then it can be improved to a theoretical estimate of $O(n^{2.376})$ which in reality is usually close to $O(n^{2.81})$.

The second approach was proposed in the work [2] by D. Aingworth et al. This approach assumes that the set of vertices is divided into two subsets; the first contains vertices with high degrees and the second one with low degrees. This heuristic separation allows to start BFS algorithm rationally and reduce the asymptotics to $O(m\sqrt{n})$. Further, this estimate was improved in the works [7–9]. However, all these algorithms either impose significant restrictions on graph sizes or use special data structures, which makes their practical implementation difficult. Even with these improvements, exact algorithms show good time characteristics only on graphs with small dimensions.

These methods can be well applied to small graphs. However, real-life graphs, that are of most interest for research, contain tens of thousands of nodes. As a result, exact algorithms cannot be applied to solve similar tasks due to unacceptable time costs. In this situation heuristic algorithms that give better temporal results may be the most suitable, but they do not guarantee the exact solution.

This work presents Ball-Shrinking Search Algorithm [10] that is a genetic algorithm aimed to find the central vertices in a graph. This algorithm was compared with some exact algorithms and with the genetic algorithm from the article [4] and proved to be promising. The paper examines the influence of parameters of a genetic algorithm, such as population size, probability of selection and mutation, on its performance.

The problem of choosing optimal parameters for heuristic and genetic algorithms was addressed in many papers, e.g. [11–19]

2. Ball-Shrinking Search Algorithm

In order to find the exact solution to the problem, it is necessary to find the eccentricity of each vertex using the breadth-first search algorithm, and then to select the vertex whose eccentricity is minimal. Obviously, this approach is not acceptable for large graphs. The main idea of the algorithm is as follows. At each iteration of the proposed genetic algorithm, we find eccentricities only for a small number of vertices (individuals of the population), the convex hull of which we can conditionally call a ball. In this case, we assume that the central vertex is inside this ball.

The current population can be interpreted as a set of spheres each with a center at its own vertex from population. At the intersection of the spheres there lies a central vertex of the graph. In this regard, one can consider a pair of vertices from the population and find the shortest path between them using the BFS. After that, as a result of crossover operation for two individuals, a random vertex lying on the path is selected. It is supposed that we approach a vertex with the optimal eccentricity with each iteration (figure 1).

As a mutation process, an approach is chosen in which in order to change an existing population, the vertex is replaced by a random one from the set of its neighbors with a given probability $p_m$.

Using the selection and crossover operators, we reduce the size of this ball towards the central vertex. Meanwhile, the mutation operator pushes the process of evolution towards the search for
Figure 1. The main algorithm idea. The left figure shows the possible location of the population at the beginning of the algorithm, and the right shows the location of the population after several iterations (yellow vertex – center vertex, green color – population).

The algorithm pseudocode is presented in Algorithm 1.

Full C++ code is available on Github: https://github.com/Alexflames/bssa-genetic.

To visualize the work of the proposed algorithm, we present one of the graphs that was generated with use of random geometric model (figure 2).

Figure 2. Algorithm steps (yellow color – center vertex, red color – population, dark blue color – vertex in population that represents the right solution)
begin
  Data: Graph $G$
  Result: Vertex $c$, which is center of graph $G$
  for $i := 1$ to $\text{populationSize}$ do
    population[$i$] ← random vertex $\in G$;
  end
  for $i \leftarrow 1$ to $\text{iterationNumber}$ do
    Crossover()
    Mutation()
    Selection()
  end
  $c.eccentricity \leftarrow \infty$
  for $v \in \text{population}$ do
    if $v.eccentricity < c.eccentricity$ then
      $c \leftarrow v$
    end
  end
end

Function Mutation()
  Data: Population on current algorithm step
  Result: Population after applying a mutation operator to each individual
  for $i \leftarrow 1$ to $\text{populationSize}$ do
    if random(0, 1) $< p_m$ then
      neighbours ← population[$i$].getNeighbours
      population[$i$] ← random vertex from neighbours
    end
  end
end

Function Crossover()
  Data: Population on current algorithm step
  Result: Population after crossover
  for $i \leftarrow 1$ to $\text{populationSize}$ do
    if random(0, 1) $< p_c$ then
      $u \leftarrow$ random vertex from population
      $v \leftarrow$ random vertex from population
      path ← $G$.pathBetween($u, v$
      population.insert(population[path.size/2])
    end
  end
end

Function Selection()
  Data: Population on current algorithm step
  Result: Population for next algorithm step
  for $i \leftarrow 1$ to $\text{populationSize}$ do
    eccentricity[$i$] ← population[$i$].eccentricity
  end
  for $i \leftarrow 1$ to $\text{populationSize}$ do
    probability[$i$] ← number from $[0, 1]$ based on eccentricity value
  end
  for $i \leftarrow 1$ to $\text{populationSize}$ do
    nextPopulation[$i$] ← $v$ from population using probability
  end
end

Algorithm 1: The main steps of the proposed algorithm
3. Selection of optimal algorithm parameters

The main parameters of the genetic algorithm are the probability of a mutation $p_m$, probability crossover $p_c$ and a population size $N$. In this section we investigate the effect of these parameters on the performance of the algorithm.

In [10] the algorithm took the values $p_c = 0.7$ and $p_m = 0.1$ and $N = 20$ for these parameters. With these values, the algorithm showed quite promising results.

The running time of the algorithm and the accuracy of the solution can be considered as performance characteristics. The nature of genetic algorithms is that any changes to these parameters can significantly affect both characteristics.

A series of experiments were conducted to find parameters that are best fit for the proposed algorithm. The results are described below.

All experiments were performed on a computer with AMD A8-7410 2.20 GHz CPU and 6 GB RAM. All algorithms were implemented in the programming language C++. The algorithm was run 100 times with each set of parameters. The running time of the algorithm is calculated as an average between them. Two algorithms by D. Aingworth [2] and R. Seidel [1] are used in order to find out the exact center vertex. The results are compared with them to calculate the percentage of errors in each set of parameters.

In this article, we present the results of the algorithm for random graphs $G = (V, E)$, in which the number of vertices and the number of edges were assumed to be $|V| = 2500$, $|E| = 90268$ and $|V| = 5000$, $|E| = 358553$.

First of all, in order to find out the optimal values for the parameters $p_c$ and $p_m$, we measured the running time of the algorithm with enumeration of $p_c$ and $p_m$ in increments of 0.1.

The first series of experiments was performed with constant population size $N = 20$.

The running time of the algorithm on the geometric random graph is presented in tables 1 and 2.

| $p_m$ | 0.0  | 0.1  | 0.2  | 0.3  | 0.4  | 0.5  | 0.6  | 0.7  | 0.8  | 0.9  | 1.0  |
|-------|------|------|------|------|------|------|------|------|------|------|------|
| 0.0   | 0.06 | 0.11 | 0.13 | 0.15 | 0.17 | 0.19 | 0.20 | 0.29 | 0.28 | 0.27 | 0.31 |
| 0.1   | 0.18 | 0.26 | 0.27 | 0.33 | 0.32 | 0.36 | 0.36 | 0.41 | 0.40 | 0.40 | 0.40 |
| 0.2   | 0.29 | 0.35 | 0.52 | 0.55 | 0.52 | 0.54 | 0.49 | 0.54 | 0.56 | 0.54 | 0.50 |
| 0.3   | 0.45 | 0.50 | 0.53 | 0.57 | 0.60 | 0.54 | 0.60 | 0.58 | 0.65 | 0.68 | 0.66 |
| 0.4   | 0.51 | 0.53 | 0.64 | 0.71 | 0.69 | 0.62 | 0.64 | 0.66 | 0.70 | 0.75 | 0.78 |
| 0.5   | 0.62 | 0.67 | 0.74 | 0.79 | 0.78 | 0.84 | 0.87 | 0.82 | 0.80 | 0.82 | 0.93 |
| 0.6   | 0.60 | 0.77 | 0.77 | 0.96 | 0.87 | 0.86 | 0.88 | 0.93 | 1.01 | 1.04 | 1.14 |
| 0.7   | 0.86 | 0.96 | 1.04 | 1.09 | 1.13 | 1.09 | 1.00 | 1.15 | 1.03 | 1.32 | 1.00 |
| 0.8   | 0.76 | 0.85 | 0.89 | 0.93 | 0.98 | 0.96 | 1.02 | 1.19 | 1.05 | 1.08 | 1.13 |
| 0.9   | 0.82 | 0.91 | 1.01 | 1.01 | 1.23 | 1.29 | 1.30 | 1.31 | 1.18 | 1.20 | 1.10 |
| 1.0   | 0.96 | 1.08 | 1.03 | 1.22 | 1.25 | 1.30 | 1.30 | 1.26 | 1.18 | 1.32 | 1.50 |

As it can be seen from tables 1 and 2, the algorithm running time increases as it grows $p_c$, $p_m$. The results shown in these tables are predictable, since, by increasing the probability of an operation we increase the frequency of its execution.

A similar experiment for a percentage error measurement was carried out, the results of which are given in tables 3 and 4.

According to results from tables 3 and 4 the percentage of incorrectly found answers decreases with the increase of the values of $p_m$ and $p_c$. These results can also be called foreseeable since
Table 2. The runtime (sec.) of BSSA on geometric graph $|V| = 5000$, $|E| = 358553$, $N = 20$

| $p_m$ | 0.0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 0.00  | 0.07| 0.13| 0.17| 0.21| 0.33| 0.30| 0.39| 0.38| 0.39| 0.32| 0.36|
| 0.10  | 0.22| 0.31| 0.36| 0.39| 0.42| 0.44| 0.47| 0.53| 0.59| 0.55| 0.57|
| 0.20  | 0.44| 0.54| 0.60| 0.64| 0.76| 0.73| 0.64| 0.75| 0.67| 0.69| 0.68|
| 0.30  | 0.52| 0.77| 0.80| 0.83| 0.84| 0.88| 0.80| 0.83| 0.85| 0.85| 0.86|
| 0.40  | 0.61| 0.74| 0.83| 1.17| 1.02| 1.12| 1.07| 1.01| 1.13| 1.20| 1.34|
| 0.50  | 0.88| 1.07| 0.96| 0.98| 1.03| 1.06| 1.08| 1.10| 1.11| 1.13| 1.29|
| 0.60  | 0.82| 0.96| 1.03| 1.16| 1.28| 1.25| 1.30| 1.31| 1.29| 1.39| 1.46|
| 0.70  | 1.17| 1.14| 1.25| 1.46| 1.36| 1.44| 1.58| 1.40| 1.46| 1.44| 1.56|
| 0.80  | 1.05| 1.18| 1.27| 1.36| 1.41| 1.43| 1.49| 1.51| 1.56| 1.54| 1.57|
| 0.90  | 1.17| 1.31| 1.42| 1.66| 1.55| 1.58| 1.61| 1.66| 1.68| 1.74| 1.63|
| 1.00  | 1.19| 1.41| 1.52| 1.73| 1.69| 1.69| 1.75| 1.72| 1.79| 1.86| 1.89|

Table 3. Error percent (%) of BSSA on geometric graph $|V| = 2500$, $|E| = 90286$, $N = 20$

| $p_m$ | 0.0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 0.0   | 81  | 54  | 32  | 52  | 42  | 52  | 59  | 53  | 62  | 60  | 61  |
| 0.1   | 48  | 35  | 19  | 28  | 33  | 28  | 32  | 23  | 39  | 31  |
| 0.2   | 46  | 27  | 29  | 20  | 21  | 25  | 21  | 26  | 21  | 17  |
| 0.3   | 44  | 23  | 18  | 17  | 20  | 18  | 15  | 14  | 17  | 21  |
| 0.4   | 32  | 28  | 19  | 22  | 9   | 16  | 21  | 16  | 18  | 30  | 16  |
| 0.5   | 32  | 16  | 14  | 19  | 20  | 16  | 22  | 14  | 14  | 9   |
| 0.6   | 30  | 18  | 12  | 21  | 11  | 12  | 15  | 9   | 12  | 20  | 18  |
| 0.7   | 32  | 17  | 20  | 14  | 13  | 10  | 19  | 12  | 22  | 8   | 12  |
| 0.8   | 33  | 22  | 20  | 14  | 21  | 15  | 21  | 16  | 15  | 14  | 19  |
| 0.9   | 24  | 18  | 24  | 15  | 23  | 17  | 19  | 14  | 20  | 11  | 14  |
| 1.0   | 32  | 16  | 13  | 14  | 12  | 15  | 16  | 14  | 7   | 13  | 9   |

an increase in the number of crossover and mutation results should lead to an increase in the variety of vertices that the algorithm operates on, in which the desired solution may turn out to be.

For a general estimation of the parameters, we introduce the function $F$, which allows us to find the optimal parameters $p_m$ and $p_c$:

$$F(p_m, p_c) = \alpha \text{time}(p_m, p_c) + \beta \text{error}(p_m, p_c).$$  \hspace{1cm} (1)$$

The values of the function for $\alpha = 0.3$ and $\beta = 0.7$ with normalized runtime and percentage of errors are presented in tables 5 and 6.

As it can be seen from table 5, $F$ reaches its minimum at $p_m = 0.4$ and $p_c = 0.4$ on corresponding types of graph. Table 6 shows that for the second type of graph, the optimal probability values are $p_m = 0.2$ and $p_c = 0.3$.

In addition, we will illustrate the influence of the parameters $p_m$ and $p_c$ on the efficiency of the algorithm.
Table 4. Error percent (%) of BSSA on geometric graph $|V| = 5000$, $|E| = 358553$, $N = 20$

| $p_m$ | 0.0  | 0.1  | 0.2  | 0.3  | 0.4  | 0.5  | 0.6  | 0.7  | 0.8  | 0.9  | 1.0  |
|-------|------|------|------|------|------|------|------|------|------|------|------|
| 0.00  | 81   | 53   | 41   | 45   | 47   | 40   | 53   | 49   | 53   | 54   | 51   |
| 0.10  | 70   | 39   | 31   | 28   | 33   | 34   | 21   | 25   | 26   | 25   | 24   |
| 0.20  | 45   | 22   | 30   | 14   | 27   | 21   | 20   | 18   | 22   | 21   | 21   |
| 0.30  | 52   | 27   | 23   | 19   | 17   | 15   | 19   | 21   | 23   | 22   | 22   |
| 0.40  | 47   | 25   | 18   | 25   | 18   | 14   | 23   | 19   | 19   | 15   | 16   |
| 0.50  | 36   | 22   | 19   | 23   | 18   | 20   | 16   | 15   | 18   | 17   | 16   |
| 0.60  | 41   | 16   | 27   | 16   | 18   | 12   | 18   | 15   | 20   | 9    | 10   |
| 0.70  | 45   | 23   | 21   | 18   | 17   | 20   | 16   | 23   | 12   | 18   | 16   |
| 0.80  | 39   | 22   | 20   | 22   | 17   | 16   | 15   | 13   | 19   | 17   | 16   |
| 0.90  | 35   | 25   | 17   | 20   | 19   | 20   | 19   | 22   | 11   | 20   | 17   |
| 1.00  | 33   | 19   | 20   | 18   | 17   | 16   | 21   | 10   | 11   | 17   |      |

Table 5. $F$ function values, $\alpha = 0.3$, $\beta = 0.7$, $|V| = 2500$, $|E| = 90268$, $N = 20$

| $p_m$ | 0.0  | 0.1  | 0.2  | 0.3  | 0.4  | 0.5  | 0.6  | 0.7  | 0.8  | 0.9  | 1.0  |
|-------|------|------|------|------|------|------|------|------|------|------|------|
| 0.0   | 0.71 | 0.49 | 0.30 | 0.48 | 0.40 | 0.49 | 0.55 | 0.52 | 0.59 | 0.57 | 0.59 |
| 0.1   | 0.45 | 0.36 | 0.22 | 0.31 | 0.35 | 0.37 | 0.32 | 0.36 | 0.28 | 0.42 | 0.35 |
| 0.2   | 0.46 | 0.31 | 0.36 | 0.28 | 0.29 | 0.32 | 0.32 | 0.32 | 0.34 | 0.29 | 0.25 |
| 0.3   | 0.47 | 0.30 | 0.26 | 0.26 | 0.29 | 0.27 | 0.25 | 0.24 | 0.26 | 0.28 | 0.32 |
| 0.4   | 0.38 | 0.35 | 0.29 | 0.33 | 0.32 | 0.26 | 0.31 | 0.27 | 0.30 | 0.41 | 0.30 |
| 0.5   | 0.40 | 0.27 | 0.27 | 0.32 | 0.33 | 0.31 | 0.37 | 0.32 | 0.28 | 0.29 | 0.26 |
| 0.6   | 0.38 | 0.31 | 0.26 | 0.37 | 0.27 | 0.28 | 0.31 | 0.26 | 0.31 | 0.38 | 0.38 |
| 0.7   | 0.45 | 0.34 | 0.38 | 0.34 | 0.34 | 0.30 | 0.36 | 0.33 | 0.40 | 0.33 | 0.30 |
| 0.8   | 0.44 | 0.36 | 0.35 | 0.31 | 0.38 | 0.32 | 0.39 | 0.38 | 0.34 | 0.34 | 0.39 |
| 0.9   | 0.37 | 0.34 | 0.41 | 0.33 | 0.44 | 0.40 | 0.42 | 0.38 | 0.41 | 0.34 | 0.34 |
| 1.0   | 0.47 | 0.35 | 0.32 | 0.36 | 0.35 | 0.39 | 0.40 | 0.37 | 0.30 | 0.38 | 0.38 |

Figure 3 reflects how percentage of errors depends on the running time of the algorithm with fewer vertices at $p_m = 0.4$ and $p_m = 0.6$.

Figure 4 shows a similar dependence for graphs with a larger number of vertices at $p_m = 0.2$.

The influence of the parameter $N$ on the efficiency of the algorithm is studied in the next series of experiments. The following experimental results were obtained for random graphs $G = (V, E)$, in which the number of vertices and the number of edges were assumed to be $|V| = 2500$, $|E| = 90268$.

It is well known, that the effectiveness of genetic algorithms depends on the size of the population $N$. On one hand, an increase in $N$ will lead to an increase in the run time of the algorithm, which is confirmed by figure 5.

On the other hand, an increase in this parameter will lead to a greater variety of individuals under consideration, and therefore to the probability of finding an accurate solution more quickly. This trend is shown in figure 6. It can be seen that with the increase of running time the percentage of error decreases exponentially.
Table 6. $F$ function values, $\alpha = 0.3$, $\beta = 0.7$, $|V| = 5000$, $|E| = 35853$, $N = 20$

| $p_m$ | 0.0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 0.00  | 0.71| 0.48| 0.38| 0.42| 0.46| 0.49| 0.52| 0.52| 0.52| 0.50|     |
| 0.10  | 0.64| 0.39| 0.33| 0.30| 0.35| 0.36| 0.26| 0.30| 0.32| 0.30| 0.30|
| 0.20  | 0.46| 0.28| 0.35| 0.22| 0.35| 0.30| 0.27| 0.27| 0.34| 0.30| 0.29|
| 0.30  | 0.53| 0.36| 0.33| 0.30| 0.28| 0.27| 0.29| 0.31| 0.32| 0.33| 0.33|
| 0.40  | 0.50| 0.33| 0.29| 0.40| 0.32| 0.30| 0.37| 0.32| 0.34| 0.32| 0.35|
| 0.50  | 0.45| 0.36| 0.32| 0.35| 0.32| 0.34| 0.31| 0.30| 0.33| 0.33| 0.34|
| 0.60  | 0.48| 0.29| 0.40| 0.32| 0.36| 0.30| 0.36| 0.34| 0.38| 0.30| 0.32|
| 0.70  | 0.57| 0.38| 0.38| 0.39| 0.36| 0.40| 0.39| 0.42| 0.34| 0.38| 0.39|
| 0.80  | 0.50| 0.38| 0.37| 0.41| 0.37| 0.37| 0.37| 0.36| 0.41| 0.40|     |
| 0.90  | 0.49| 0.42| 0.37| 0.44| 0.41| 0.42| 0.42| 0.45| 0.36| 0.45| 0.41|
| 1.00  | 0.47| 0.39| 0.41| 0.43| 0.42| 0.41| 0.43| 0.45| 0.37| 0.39| 0.45|

Figure 3. Normalized values of run-time and error percent, $|V| = 2500$, $|E| = 90268$, $N = 20$: (a) $p_m = 0.4$; (b) $p_m = 0.6$

To select a proper parameter $N$, we introduce the function $F_1$, similar to function $F$:

$$F_1(N) = \alpha time(N) + \beta error(N).$$

Choosing $\alpha$ and $\beta$ allows one to visually select the parameter $N$ in an optimal way. Figure 7 shows different behavior of the function for $\alpha = 0.3$, $\beta = 0.7$, and $\alpha = 0.5$, $\beta = 0.5$.

From figure 7(b), we can conclude that the optimal value of parameter $N$ for this type of graph is $N = 30$.

4. Empirical results

Based on experimental results, an example of which is given in the previous section, the parameters $p_m = 0.3$, $p_c = 0.4$, $N = 30$ were selected as optimal for our algorithm. These parameters were used for all following test runs.

To test the effectiveness of the algorithm for the chosen parameters, another (N4N) genetic algorithm was used, which is described in [4]. It has similar approaches to solving the considered problem but differs in the rules of mutation, crossover operator and population selection.
Algorithms were compared on graphs generated with the use of three random graph models
- the well-known Barabasi-Albert model [5],
- the random geometric graph [6]
- the Erdos-Renyi model [3].

We choose parameter $m = 2$ for the BA graph, $r = 0.1$ in geometric graph and we set $p = 1\%$ in Erdos-Renyi model.

For both genetic algorithms, the values of running time and the accuracy were found. Both algorithms were run 100 times on each test, which allowed us to obtain the average running time and percentage of errors – the ratio of the number of correctly found solutions to the total number of iterations. The results of the experiments are given in tables 7, 8 and 9.

Also, the running time of the proposed algorithm was compared with running time of Aingworth algorithm, that is one of the well-known exact algorithms. The results of these experiments can be seen in figure 8.

Furthermore, the algorithm is applied on real graphs of larger sizes and compared to the
Figure 7. Dependence of minimum $F_1$ function values on the population size $N$, with the parameters $p_m = 0.4$, $p_c = 0.4$: (a) $\alpha = 0.3$, $\beta = 0.7$; (b) $\alpha = 0.5$, $\beta = 0.5$.

Table 7. Running time and percentage of algorithms errors on Erdos-Renyi random graphs. BSSA – Ball-Shrinking Search Algorithm, N4NA – N4N Algorithm, AA – Aingworth Algorithm

| Graph size | Time, sec. | Error, % |
|------------|------------|----------|
| $|V|$ | $|E|$ | BSSA | N4NA | AA | BSSA | N4NA |
| 500 | 3572 | 0.03 | 0.39 | 0.07 | 42.0 | 40.0 |
| 1000 | 14202 | 0.13 | 0.77 | 0.56 | 10.0 | 50.0 |
| 1500 | 31861 | 0.23 | 1.44 | 1.13 | 5.0 | 62.0 |
| 2000 | 57438 | 0.32 | 1.92 | 1.70 | 11.0 | 48.0 |
| 2500 | 90268 | 0.43 | 2.68 | 3.20 | 0.0 | 30.0 |
| 5000 | 358553 | 1.21 | 8.61 | 18.45 | 0.0 | 0.0 |
| 10000 | 1439255 | 5.45 | 26.0 | 41.47 | 0.0 | 0.0 |

Table 8. Running time and percentage of algorithms errors on BA graphs. BSSA – Ball-Shrinking Search Algorithm, N4NA – N4N Algorithm, AA – Aingworth Algorithm

| Graph size | Time, sec. | Error, % |
|------------|------------|----------|
| $|V|$ | $|E|$ | BSSA | N4NA | AA | BSSA | N4NA |
| 500 | 996 | 0.02 | 0.29 | 0.08 | 20.0 | 0.0 |
| 1000 | 1996 | 0.08 | 0.68 | 0.36 | 12.0 | 0.0 |
| 1500 | 2996 | 0.13 | 1.24 | 0.98 | 0.0 | 0.0 |
| 2000 | 3996 | 0.17 | 1.67 | 1.68 | 0.0 | 0.0 |
| 2500 | 4996 | 0.24 | 2.18 | 3.22 | 0.0 | 0.0 |
| 5000 | 9996 | 0.54 | 8.28 | 14.17 | 0.0 | 0.0 |
| 10000 | 19996 | 4.96 | 15.8 | 24.56 | 0.0 | 0.0 |

exact solution obtained on them. Real graphs are taken from Stanford Large Network Dataset Collection (SNAP) [20]. Our algorithm shows little error and yields fast results while requiring acceptable amount of memory due to ignoring vertices that are less likely to be central. The results are given in table 10.
Table 9. Running time and percentage of algorithms errors on random geometric graphs. BSSA – Ball-Shrinking Search Algorithm, N4NA – N4N Algorithm, AA – Aingworth Algorithm

| Graph size | Time, sec. | Error, % |
|------------|------------|----------|
| | BSSA | N4NA | AA | BSSA | N4NA |
| 500 | 3572 | 0.02 | 0.39 | 0.07 | 44.0 | 40.0 |
| 1000 | 14202 | 0.08 | 0.77 | 0.56 | 8.0 | 50.0 |
| 1500 | 31861 | 0.16 | 1.44 | 1.13 | 16.0 | 62.0 |
| 2000 | 57438 | 0.25 | 1.92 | 1.70 | 13.0 | 48.0 |
| 2500 | 90268 | 0.39 | 2.68 | 3.20 | 3.0 | 30.0 |
| 5000 | 358553 | 1.10 | 8.61 | 18.45 | 0.0 | 0.0 |
| 10000 | 1439255 | 10.03 | 26.0 | 41.47 | 0.0 | 0.0 |

Figure 8. Plots of the running time versus graph sizes (blue color – trivial \(O(nm + n^2)\) algorithm, red – algorithm with improved asymptotic behavior of \(O(m\sqrt{n})\), brown – genetic algorithm): (a) Erdos-Renyi model \(p = 1\%\), (b) Barabasi-Albert model \(m = 2\), (c) geometric random graph \(r = 0.1\).

Table 10. Running time and percentage of algorithms errors on real graphs. BSSA – Ball-Shrinking Search Algorithm, Trivial – Trivial Algorithm. (1) – gemsec-Facebook(artists network) [21], (2) – MUSAE GitHub Social Network [22].

| Graph type | Graph size | Time, sec. | Error, % |
|------------|------------|------------|----------|
| | \(|V|\) | \(|E|\) | BSSA | Trivial | BSSA | Trivial |
| (1) | 50515 | 819306 | 81.35 | 2437.46 | 8.0 | 0.0 |
| (2) | 37700 | 289003 | 22.09 | 268.84 | 0.0 | 0.0 |

From the obtained results it can be seen that the proposed algorithm works faster than the Aingworth algorithm and the N4N algorithm. In this case, the algorithm gives a significant percentage of error only on graphs of relatively small size, and since the time costs of exact algorithms are insignificant it is not advisable to use this approach. However, with the increase of graph size, the percentage of incorrect answers tends to minimize.
5. Conclusions
This work investigates the main parameters of the genetic algorithm that was proposed in the work [10], such as mutation probability $p_m$, crossover probability $p_c$, and population size $N$. A series of experiments were conducted that showed the strengths and weaknesses of our algorithm. Based on these results, it can be concluded that the running time of the algorithm increases significantly with increasing parameters $p_m$, $p_c$. However, the percentage of incorrect answers is significantly reduced. In addition, the percentage of errors is affected by population size. In this regard, it can be said that parameters should be selected based on time constraints and requirements for the percentage of errors.

Acknowledgments
This work was supported by the Ministry of Science and education of the Russian Federation in the framework of the basic part of the scientific research state task, project FSRR-2020-0006.

References
[1] Seidel R 1995 On the all-pairs-shortest-path problem in unweighted undirected graphs J. Comp. Syst. Sci. 51 400–3
[2] Aingworth D, Chekuri C, Indyk P and Motwani R 1999 Fast estimation of diameter and shortest paths (without matrix multiplication) SIAM J. Comput 28 1167–1181
[3] Erdös P and Rényi A 1959 On random graphs I. Publ. Math. Debr. 6 290
[4] Alkhalifah Y and Wainwright R 2004 A genetic algorithm applied to graph problems involving subsets of vertices vol.1 Portland pp 303–8
[5] Réka A and Barabasi A 2002 Statistical mechanics of complex networks Rev. Mod. Phys. 74.1 47–97.
[6] Gilbert E N 1961 Random plane networks J Soc. Ind. and App. Math. 9 533–43
[7] Berman P and Kasiviswanathan S P 2007 Faster Approximation of Distances in Graphs Algorithms and Data Structures Lecture Notes in Computer Science (vol 4619) ed F Dehne and J R Sack et al (Berlin: Springer) pp 541–52
[8] Timothy M C 2012 All-pairs shortest paths for unweighted undirected graphs in o(mn) time ACM Trans. Alg. 8 1–17
[9] Roditty L and Williams V V 2013 Fast Approximation Algorithms for the Diameter and Radius of Sparse Graphs Proc. of the Annual ACM Symposium on Theory of Computing (New York: Association for Computing Machinery) pp 515–24
[10] Vlasov A, Khomchenko A, Faizliev A and Mironov S 2019 Ball-Shrinking Genetic Search Algorithm for Finding Central Vertices in Graphs Proc of the Fourth Workshop on Computer Modelling in Decision Making (Saratov) pp 94–7
[11] Grishina N, Lucas C A and Date P 2016 Prospect theory based portfolio optimisation: an empirical study and analysis using intelligent algorithms Quant. Fin. 17 353–67
[12] Ho C W, Lee K H and Leung K S 1999 A genetic algorithm based on mutation and crossover with adaptive probabilities Proc. of the 1999 Congress on Evolutionary Computation-CEC99 vol 1 (IEEE) pp 768–75
[13] Angelova M and Pencheva T 2011 Tuning genetic algorithm parameters to improve convergence time Int. J. Chem. Eng. 2011 1–7
[14] Stanhope S A and Daida J M 1998 Optimal mutation and crossover rates for a genetic algorithm operating in a dynamic environment Lect. Not. Comp. Sci. 1447 693–702
[15] Hassanat A, Almohammedi K, Alkafaween E, Abunasas E, Hammouri A and Prasath V B S 2019 Choosing mutation and crossover ratios for genetic algorithms—a review with a new dynamic approach Inf. 10 390
[16] Back T 1993 Optimal Mutation Rates in Genetic Search Proc. of the 5th Int. Conf. on Genetic Algorithms (San Francisco, CA: Morgan Kaufmann Publishers Inc) pp 2–8
[17] Lobo F G, Lima C F and Michalewicz Z 2007 Parameter Setting in Evolutionary Algorithms (Berlin: Springer)
[18] Eiben A and Smit S K 2012 Evolutionary Algorithm Parameters and Methods to Tune Them (Berlin: Springer) 15–36
[19] Alander J T 1992 On optimal population size of genetic algorithms CompEuro 1992 Proc. Computer Systems and Software Engineering (IEEE) pp 65–70
[20] Leskovec J and Krevl A 2014 SNAP datasets: stanford large network dataset collection Comp. Sci
[21] Rozemberczki B, Davies R, Sarkar R and Sutton C 2018 GEMSEC: graph embedding with self clustering ASONAM 2019
[22] Rozemberczki B, Allen C and Sarkar R 2019 Multi-scale attributed node embedding Comp. Sci. Math.