Atlantis Highlights in Computer Sciences, volume 2
Fourth Workshop on Computer Modelling in Decision Making (CMDM 2019)

Ball-Shrinking Genetic Search Algorithm for Finding Central Vertices in Graphs

Andrew Vlasov, Andrew Khomchenko, Alexey Faizliev*[0000−0001−6442−4361], Sergei Mironov[0000−0003−3699−5006]
Saratov State University, Saratov, Russian Federation
*faizlievar1983@mail.ru

***Abstract*—The paper proposes a genetic algorithm (GA) for finding central vertices in a graph. The algorithm uses a different approach to the method presentation of the solution and describes a new look at the crossover process of GA. The algorithm was compared with existing exact and other genetic algorithms on various random graphs. Empirical results show that this approach can be used in applications and compete with existing algorithms.**

***Index Terms*—Genetic algorithm, Graph models, Central node, Central vertices, Large graphs**

I. Introduction

Recently, with the development of modern technologies and the digitization of modern society, large volumes of data have been of greatest interest for research. These studies allow us to identify previously unstudied features of the interaction of large groups of interacting objects. Moreover, this knowledge can represent not only theoretical benefits, but also enormous practical and often commercial benefits. Obviously, for such studies it is necessary to have appropriate algorithms that could process large amounts of data in a reasonable 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 radius of a graph and central vertices is one of the important problems in theoretical and practical applications. Let us consider 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. To find the center vertex in the graph, one can use the concept of vertex eccentricity, i.e. the distance from the vertex to the most distant vertex. Then the radius of the graph can be described as the minimum of the eccentricities of all the vertices. At the same time, the nodes on which this minimum is reached are usually called central. It is also obvious that the graph in question must be connected, since the absence of a path between any pair of vertices will mean that the distance between these vertices is infinitely large. This task often arises in optimization problems in computer networks and in transport routing problems.

Different exact algorithms were proposed and developed to find the solution to the problem of finding graph central nodes. Often, it is considered together with the problem of calculating all-pairs shortest path in graph for unweighted graphs. It should be noted that one can solve this problem in a trivial way by running breadth-first search (BFS) from each vertex. This trivial algorithm has a time estimate $O(nm)$ that for large m equals $O(n^3)$. All existing algorithms, that are aimed at obtaining the exact solution, attempt to improve this asymptotic estimate. In general, there are two approaches for solving this problem. The first one is proposed in [1] and is based on matrix multiplication, since there have been developed a fast matrix multiplication algorithms that give a theoretical estimate $O(n^{2.376})$ or more practiced $O(n^{2.81})$.

Another approach was developed in [2] by D. Aingworth et al. In this case, the idea of dividing vertices into a set of vertices with a high degree and low degree is used. Using this approach, the time estimate can be improved and the asymptotics $O(m\sqrt{n})$ can be obtained. This work gave impetus to further research using a similar method and there are a number of algorithms [3]–[5] that improve this estimate. These algorithms use special data structures or assume low density of graphs, i.e. the graphs are sparse.

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. The most suitable in this situation may be heuristic algorithms that give better temporal results, but they do not guarantee the obtaining of the exact solution. For example, the work [6] describes an genetic algorihtm that allows to solve a number of graph problems. This genetic algorithm uses the classical stages of the genetic approach, such us natural selection, crossing and mutation. Moreover, the paper considers the application of this genetic algorithm for solving the k-center problem. This problem is similar to the problem of finding center vertex, only instead of finding one vertex with a minimum eccentricity, it is necessary to find k vertex whose sum of eccentricity is minimal. Easy to notice if k is one, then this problem coincides with the problem of finding center vertex.

In our work, we present a genetic algorithm that solves the problem of finding the central vertices and the radius of a graph. At the same time, in comparison with some well-known algorithms, the proposed algorithm has a better computational cost.

II. Ball-Shrinking Search Algorithm

Genetic and evolutionary-based algorithms are well-known for solving various optimization problems. The basic idea of the GA was introduced by Holland [7]. The algorithm uses genetics processes that ensure the evolutionary development of living organisms. According to the proposed approach, the solution is represented by a certain set of genes and the main stages of the algorithm are the mutation process, crossover and natural selection. In our algorithm, all these processes are implemented taking into account the problem in question.

The main idea of the algorithm is as follows. 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. Therefore, at each iteration of the proposed genetic algorithm, we find the 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. Using the selection and crossover operators, we reduce the size of this ball towards the central vertex. On the other hand, the mutation operator pushes the process of evolution towards the search for a global rather than a local solution. As a rule, the diameter of the ball containing individuals of the current population decreases with each iteration, i.e. balls shrink. Therefore, we will call such an algorithm ball-shrinking algorithm.

1) Problem Representation: The population for the genetic algorithm is a subset V' of vertices of graph $G = (V, E)$, $V' \subset V$. In the language of GA, this means that each individual in the population is a vertex of the graph. The initial population is generated as a set of random vertices.

2) Fitness Function: The most natural way to assess the quality of the solution obtained in the framework of the problem in question is the value of vertex eccentricity that is found using a breadth first search (BFS). In this case, natural selection gives priority to the vertices with a lower eccentricity.

3) Mutation: As a mutation process, we chose an approach in which 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.

4) Crossover: As a crossover operator, we used the following heuristics. Let us consider the current population, as mentioned earlier it can be interpreted as a set of spheres with centers at these vertices, at the intersection of which there is 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 descendant from 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. For a better understanding of the crossover idea you can see Figures 1.

The pseudocode is presented in Algorithm 1.

```
Data: Graph $G$
Result: Vertex $c$, which is center of graph $G$
begin
    for $i := 1$ to $populationSize$ do
        $population[i] \leftarrow random\ vertex\ \in G$;
    end
    for $i \leftarrow 1$ to $iterationNumber$ do
        $Crossover()$
        $Mutation()$
        $Selection()$
    end
    $c.eccentricity \leftarrow \infty$
    for $v \in population$ 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 $populationSize$ do
        if $randomValue < mutationProbability$
         then
            $neighbours \leftarrow$
             $popilation[i].getNeighbours$
            $population[i] \leftarrow$
             $random\ vertex\ from\ neighbours$
        end
    end
end
Function Crossover()
    Data: Population on current algorithm step
    Result: Population after crossover
    for $i \leftarrow 1$ to $populationSize$ do
        if $randomValue < crossPopulation$ then
            $u \leftarrow random\ vertex\ from\ popualtion$
            $v \leftarrow random\ vertex\ from\ population$
            $path \leftarrow G.pathBetween(u, v)$
        end
    end
end
Function Selection()
    Data: Population on current algorithm step
    Result: Population for next algorithm step
    for $i \leftarrow 1$ to $populationSize$ do
        $eccentricity[i] \leftarrow population[i].eccentricity$
    end
    for $i \leftarrow 1$ to $populationSize$ do
        $probability[i] \leftarrow number\ from\ [0, 1]$
    end
    for $i \leftarrow 1$ to $populationNumber$ do
        $nextPopulation[i] \leftarrow$
         $v\ from\ population\ with\ using\ probability$
    end
end
```

Algorithm 1: The main steps of the proposed algorithm

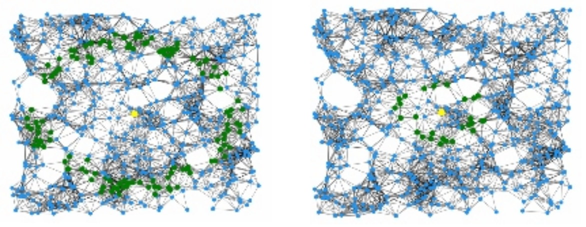

Fig. 1. The main algorithm idea. The left figure shows the possible location of the population at the beginning of the algorithm, and the right shows the location of the population after several iterations (yellow vertex – center vertex, green color – population)

III. Empirical Results

All algorithms 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 ++. We set parameters as follows: the population size is 50, the crossover probability is 0.7, the mutation probability is 0.1 and the number of evolution steps is 20. This parameters were used for all our test runs.

To compare the time costs, another genetic algorithm was used called N4N algorithm, described in [6], which has similar approaches to solving the considered problem, but differs by the mutation, crossover operator and population representation.

In addition, since the algorithm assumes a certain percentage of error, two exact algorithms by D. Aingworth [2] and R. Seidel [1] were implemented to determine it. Algorithms were compared on graphs generated with use of three random graph models

- the well-known Barabasi-Albert model [8],
- the random geometric graph [9]
- the Erdos-Renyi model [10].

For the BA graph we choose parameter $m = 2$, in a geometric random graph we choose $r = 0.1$ and in Erdos-Renyi model we set $p = 1\%$. For the 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 I, II and III.

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).

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 3.

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, where it is not advisable to use this approach, since the time costs of exact algorithms are insignificant. However, with increase of graph size, the percentage of incorrect answers tends to minimize.

TABLE I
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	0.11	0.39	0.07	38.0	40.0
1000	14202	0.31	0.77	0.56	21.0	50.0
1500	31861	0.71	1.44	1.13	13.0	62.0
2000	57438	1.20	1.92	1.70	8.0	48.0
2500	90268	1.76	2.68	3.20	4.0	30.0
5000	358553	4.48	8.61	18.45	0.0	0.0
10000	1439255	13.54	26.0	41.47	0.0	0.0

TABLE II
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	0.07	0.29	0.08	16.0	0.0
1000	1996	0.18	0.68	0.36	12.0	0.0
1500	2996	0.37	1.24	0.98	4.0	0.0
2000	3996	0.55	1.67	1.68	1.0	0.0
2500	4996	0.69	2.18	3.22	0.0	0.0
5000	9996	1.84	8.28	14.17	0.0	0.0
10000	19996	3.90	15.8	24.56	0.0	0.0

TABLE III
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, %	
$\|V\|$	$\|E\|$	BSSA	N4NA	AA	BSSA	N4NA
500	3572	0.11	0.39	0.07	38.0	40.0
1000	14202	0.31	0.77	0.56	21.0	50.0
1500	31861	0.71	1.44	1.13	13.0	62.0
2000	57438	1.20	1.92	1.70	8.0	48.0
2500	90268	1.76	2.68	3.20	4.0	30.0
5000	358553	4.48	8.61	18.45	0.0	0.0
10000	1439255	13.54	26.0	41.47	0.0	0.0

In addition, we carried out a number of measurements on graphs taken from the DIMACS library. The results of these calculations are presented in Table IV.

The graphs on which we run the algorithms to obtain empirical results presented in Tables I, II and III are rather small. The study of the algorithm behaviour on large networks is under question and it could be carried out in the future work. The limitation of the proposed algorithm on the graph size is a consequence of the limited PC RAM on which the experiments were performed. The size of the graphs can be increased at the expense of the long-term memory of the computer. This process requires the use of additional software and, based on the data obtained, is not advisable at this stage of work, however, it may be part of further research.

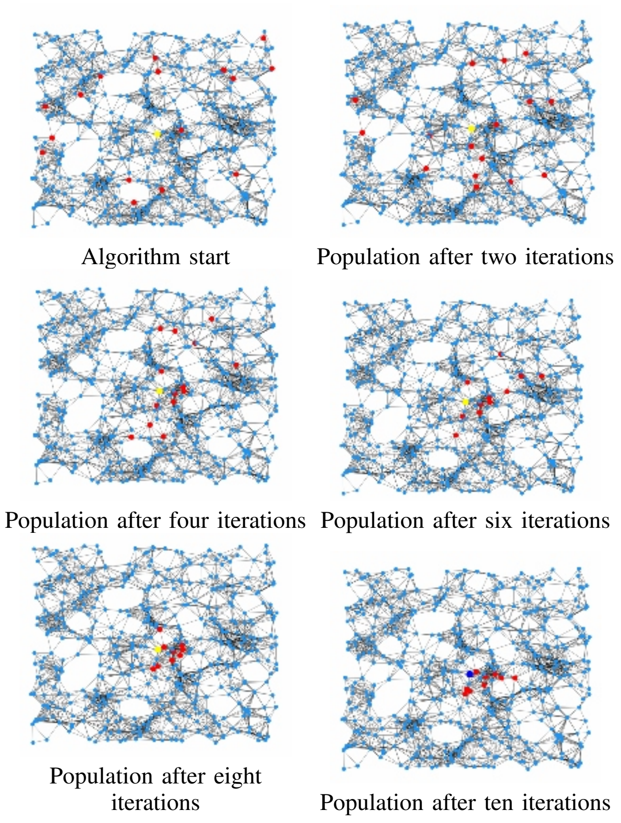
Algorithm start
Population after two iterations
Population after four iterations
Population after six iterations
Population after eight iterations
Population after ten iterations

Fig. 2. Algorithm steps (yellow color – center vertex, red color – population, dark blue color – vertex in population that represetns the right solution)

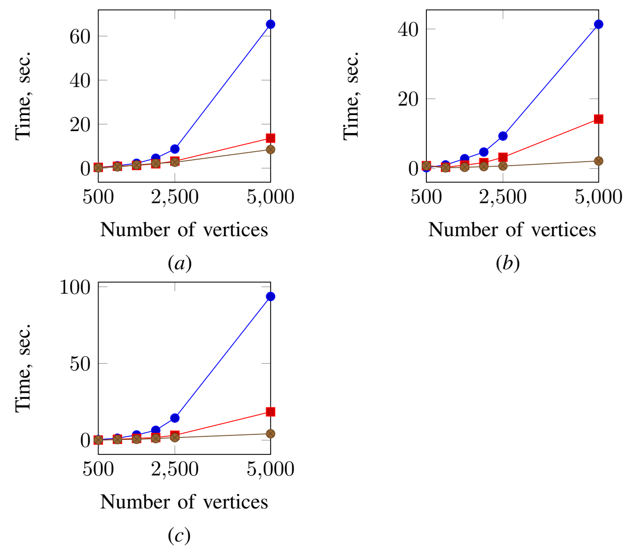

Fig. 3. Plots of the running time versus graph sizes (blue color – trivial $O(nm + n^2)$ algorithm, red – algorithm with improved asymptotic behavior of $O(m\sqrt{n})$, brow — genetic algorithm): (a) – Erdos-Renyi model $p = 1\%$, (b) – Barabasi-Albert model $m = 2$, (c) – geometric random graph $r = 0.1$

TABLE IV
RUNNING TIME OF ALGORITHMS ON THE GRAPHS FROM DIMACS LIBRARY. BSSA – BALL-SHRINKING SEARCH ALGORITHM, N4NA – N4N ALGORITHM, AA – AINGWORTH ALGORITHM, TA – TRIVIAL ALGORITHM

	Graph size		Time, sec.			Error, %
	$\lvert V\rvert$	$\lvert E\rvert$	TA	AA	BSSA	TA AA BSSA
c250.g	250	27984	0.09	0.04	0.06	0.0
c500.g	500	112372	0.91	0.31	0.25	0.0
c1000.g	1000	450079	5.33	1.26	0.97	0.0
c2000.5	2000	999836	24.49	4.16	2.32	0.0
c4000.5	4000	4000268	182.26	21.42	8.81	0.0

IV. Conclusion

We have proposed and implemented a genetic algorithm for solving the problem of finding the central vertex and radius of the graph. We tested the proposed algorithm using three random graph models and DIMACS graphs. Having obtained empirical results, we can say that the proposed algorithm may be applicable in the study of large graphs, where it is not possible to use exact algorithms due to the unacceptable time costs that they entail. At the same time, the algorithm that we propose has significantly lower time costs, with a relatively small percentage of errors.

Acknowledgments

The work was supported by Russian Foundation for Basic Research (grant 18-37-00060).

References

[1] R. Seidel, "On the all-pairs-shortest-path problem in unweighted undirected graphs," *J. Comput. Syst. Sci.*, vol. 51, no. 3, pp. 400–403, 1995.

[2] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani, "Fast estimation of diameter and shortest paths (without matrix multiplication)," *SIAM J. Comput.*, vol. 28, no. 1, pp. 1167–1181, 1999.

[3] P. Berman and S. P. Kasiviswanathan, "Faster approximation of distances in graphs," in *Algorithms and Data Structures*, F. Dehne, J.-R. Sack, and N. Zeh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[4] T. M. Chan, "All-pairs shortest paths for unweighted undirected graphs in o(mn) time," *ACM Trans. Algorithms*, vol. 8, no. 4, pp. 34:1–34:17, Oct. 2012. [Online]. Available: http://doi.acm.org/10.1145/2344422.2344424

[5] L. Roditty and V. Vassilevska Williams, "Fast approximation algorithms for the diameter and radius of sparse graphs," in *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '13. New York, NY, USA: ACM, 2013, pp. 515–524. [Online]. Available: http://doi.acm.org/10.1145/2488608.2488673

[6] Y. Alkhalifah and R. L. Wainwright, "A genetic algorithm applied to graph problems involving subsets of vertices," in *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, vol. 1, June 2004, pp. 303–308.

[7] J. Holland, *Adaption in Natural and Artificial Systems Adaption in Natural and Artificial Systems*. University of Michigan Press, 1975.

[8] R. Albert and A.-L. Barabasi, "Statistical mechanics of complex networks," *Reviews of Modern Physics*, vol. 74, no. 1, pp. 47–97, 2002.

[9] E. Gilbert, "Random plane networks," *Journal of the Society for Industrial and Applied Mathematics*, vol. 9, no. 4, pp. 533–543, 1961.

[10] P. Erdös and A. Rényi, "On random graphs I," *Publicationes Mathematicae Debrecen*, vol. 6, p. 290, 1959.