International Journal of Electrical and Computer Engineering (IJECE)
Vol. 12, No. 2, April 2022, pp. 2058~2065
ISSN: 2088-8708, DOI: 10.11591/ijece.v12i2.pp2058-2065

An analysis between exact and approximate algorithms for the k-center problem in graphs

Velin Kralev, Radoslava Kraleva, Viktor Ankov, Dimitar Chakalov
Department of Informatics, Faculty of Mathematics and Natural Sciences, South-West University "Neofit Rilski", Blagoevgrad, Bulgaria

Article Info

Article history:

Received May 16, 2021
Revised Jul 23, 2021
Accepted Aug 14, 2021

Keywords:

Computational complexity
Graph theory
Heuristic algorithms
K-center problem
Optimization problems

ABSTRACT

This research focuses on the k-center problem and its applications. Different methods for solving this problem are analyzed. The implementations of an exact algorithm and of an approximate algorithm are presented. The source code and the computation complexity of these algorithms are presented and analyzed. The multitasking mode of the operating system is taken into account considering the execution time of the algorithms. The results show that the approximate algorithm finds solutions that are not worse than two times optimal. In some case these solutions are very close to the optimal solutions, but this is true only for graphs with a smaller number of nodes. As the number of nodes in the graph increases (respectively the number of edges increases), the approximate solutions deviate from the optimal ones, but remain acceptable. These results give reason to conclude that for graphs with a small number of nodes the approximate algorithm finds comparable solutions with those founds by the exact algorithm.

Corresponding Author:

Velin Kralev
Department of Informatics, South-West University "Neofit Rilski"
66 Ivan Michailov str., 2700 Blagoevgrad, Bulgaria
Email: velin_kralev@swu.bg

1. INTRODUCTION

Graph theory is a field of computer science that has constantly been evolving in recent years [1]. Graphs are very useful data structures [2]. They are used to describe and study various relationships of real-world objects [3], [4]. Complex problems can be visually and effectively presented and solved through the graphs. For this purpose it is necessary to select the appropriate data structures and to use the appropriate algorithms [5], [6]. Typically, these problems are solved by a computer running a specific application program that executes a specific algorithm. This determines the great interest of researchers in synthesizing, analyzing and implementing various algorithms for solving the problems described by graphs. This process also includes the creation of application programs developed through the relevant programming environments and languages [7].

Finding centers in graphs is a classic NP-hard problem [8]. This problem is being actively studied, as are various algorithms for solving it [9]–[11]. Different variants and versions of the problem are described in the scientific literature [12]–[15]. Some of these variants are based on non-smooth optimization techniques [13], others are related to restricted covering problem [16] and fixed-parameter approximations in transportation networks [17], [18], as well as those applied in trees [19]. In [20] a detailed overview of the various methods and algorithms used to solve this problem is given. They are mainly used for problems related to locating different types of objects [21]–[26].

The graph G can be represented as the set that contains two other sets V and E. The set V contains the vertices (or also called nodes), and the set E contains the edges that connect some pairs of vertices. Analytically, this can be represented as G=(V, E), where V={v_1, v_2, ..., v_n} is a set of vertices, and E={e_1, e_2, ..., e_m} is a set of undirected edges. The sets V and E are finite. If each element e_k, (k=1, 2, ..., m) is an ordered pair (v, u), as v, u ∈ V, then the graph is called directed, and the set of edges is denoted by A, as the corresponding directed edges are called arcs. In this case, the vertex v is the beginning of the arc, and the vertex u is the end of the arc. If a value is mapped to each edge, then the graph is called weighted [27], [28].

Once these notations are entered, the problem of finding a center in a graph can also be defined. If a connected undirected graph G=(V, E) is given, it is necessary to find a vertex v ∈ V such that the distance from it to the farthest vertex (for example u) is minimal, i.e., distance(v, u)=minimum. In this case, the vertex v is called the center of the graph, and the distance from the vertex u to the vertex v (i.e., distance (v, u)) is called the radius of the graph [1], [29].

When the graph is connected and directed, i.e., G=(V, A), first, the length of the minimum path from vertex v to vertex u, i.e. distance(v, u)=minimum must be found. This is the shortest way to the farthest vertex of v. Then, the length of the minimum distance from all other vertices to the vertex v, i.e., distance(u, v)=minimum, for each u ∈ V must be found. Since in this case the graph is directed, it is possible that the two paths from v to u and from u to v are different (as a length and as a sequence of vertices). The vertex v is called the outer center of the graph G, and the vertex u is called the inner center of the graph G. In this case, the minimum total distance of distance(v, u)+distance(u, v) is calculated. This distance is calculated for each pair of vertices v, u. The vertex x for which this distance is minimal is the center of the graph G. The shortest paths in the graph are calculated according to Floyd's algorithm. Once these distances have been calculated, the center and the radius are determined according to the definition. When looking for more centers in the graph, for example k, it is necessary to find the set C={c_1, c_2, ..., c_k}, such that for any vertex i the longest distance from it to one of the centers is minimal. In this case, the distances obtained after generating all k-element subsets C of V are checked. There are different algorithms for solving the problem, both exact [15] and approximate [17], [30].

In the present study, a modification of the problem of finding k centers in graphs will be studied. If the set of all vertices in the graph into two disjoint subsets V and N is divided, then the elements in the set N will be nodes, and the elements in the set V will be vertices. The distribution of the elements is such that each element of the set V is located in close proximity to at least one of the elements in the set N (i.e., near one or more nodes). Proximity is determined by the epsilon threshold (e.g., 50 meters, 50 pixels, depending on the actual object being represented). In this case, the problem of finding centers in the graph is reduced to finding centers only among the elements that belong to the set N, i.e., only between nodes.

2. RESEARCH METHOD

In this section, three algorithms for solving the k-center problem will be implemented. These algorithms use both local and global data structures. The global variables NodeCount and KRadius (of Integer type) store the number of nodes in the current graph and the length of its radius, respectively. The two-dimensional FloydMatrix array (of Integer type) stores the lengths of the shortest paths between each pair of nodes in the graph. The one-dimensional arrays Centers and KCenters (of Integer type) store the combinations of the indexes of the nodes that the exact algorithm generates (the Centers array) and the indices of the centers in the graph (the KCenters array). The one-dimensional array UsedNodes array (of Boolean type) is used by the approximate algorithm to mark the nodes that have been tested for potential centers in the graph. Figure 1 shows the source code of the FindGraphCenter procedure.

```
01 procedure FindGraphCenter;
02 begin
03   var CenterIndex: Integer: = 0;
04   var MinRadius: Integer: = MaxInt;
05   for var FRow: Integer: = 1 to NodeCount do begin
06     var MaxRadius: Integer := FloydMatrix[FRow, 1];
07     for var FCol: Integer := 1 to NodeCount do begin
08       if (FRow <> FCol) and (FloydMatrix[FRow, FCol] > MaxRadius) then
09         MaxRadius := FloydMatrix[FRow, FCol];
10     end;
11     if (MaxRadius < MinRadius) then begin
12       MinRadius := MaxRadius; CenterIndex := FRow; end;
13   end;
14 end;
```

Figure 1. Source code of the FindGraphCenter procedure

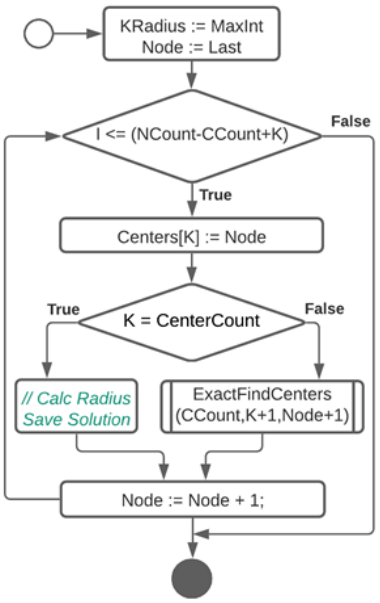
The FindGraphCenter procedure does not receive any input parameters because it is designed to search only one center in a graph. This procedure uses the global variable NodeCount (of type Integer) and the global two-dimensional array FloydMatrix (containing elements also of type Integer). The Floyd matrix contains the lengths of the shortest paths between each pair of nodes in a particular graph. The three local variables of type Integer-CenterIndex, MinRadius and MaxRadius (lines 03, 04, and 06) are declared in the body of the procedure FindGraphCenter. The variable CenterIndex stored index of the node fulfills the condition for the center of the graph. The variables MinRadius and MaxRadius are used to store the lengths of the smallest and largest radiuses found for each node in the graph.

The search for the center of the graph is performed by iterating the Floyd matrix, which is realized by two nested loops - one for iterating the rows (line 05) and one for iterating the columns (line 07). Initially, the MinRadius variable is initialized with a value equal to the MaxInt constant (i.e., a value of 2147483647). This value is the largest possible value for a 32-bit Integer type (representing infinity in this case). The FindGraphCenter procedure checks each cell in the Floyd matrix (which is not a loop in the graph, i.e., a cell of the type FRow=FCol) comparing its value with the current value of the local variable MaxRadius. Thus, the FindGraphCenter procedure finds the farthest node from the current one (indicated by the value of the FRow variable) and stores this value in the MaxRadius variable (lines 07-09). If the value of the MaxRadius variable is less than the value of the MinRadius variable, then the MaxRadius value is copied to the MinRadius variable and the value of the FRow variable is stored in the CenterIndex variable. In this case, the variable FRow stores the current index of a node in the graph at which a smaller radius was found (lines 11-12). Thus, after performing the FindGraphCenter procedure, the index of the node, which is the center of the graph, will be stored in the CenterIndex variable, and the value of the radius of the graph will be stored in the MinRadius variable. The complexity of this process is quadratic and is determined by the two nested loops that traverse the FloydMatrix matrix in rows and columns. The number of these rows and columns is determined by the value of the global variable NodeCount. It represents the number of nodes in each graph.

Figure 2(a) shows the source code and Figure 2(b) the flowchart of the ExactFindCenters recursive procedure. This procedure generates all possible combinations (without repetition) of NodeCount elements of k-th class. NodeCount is the number of nodes in the analyzed graph, and k is the number of searched centers.

```
01 procedure ExactFindCenters(CenterCount,K,Last:Integer);
02 begin
03   KRadius := MaxInt;
04   for Node := Last to (NodeCount-CenterCount+K) do
05     begin Centers[K] := Node;
06     if (K = CenterCount) then begin
07       var MinRadius: Integer := 0;
08       for var FCol: Integer := 1 to NodeCount do begin
09         var TempRadius: Integer := MaxInt;
10         for var I: Integer := 1 to CenterCount do
11           if(FloydMatrix[Centers[I]][FCol]<TempRadius)then
12             TempRadius := FloydMatrix[Centers[I]][FCol];
13         if(MinRadius<TmpRadius)then MinRadius:=TmpRadius;
14       end;
15       if (MinRadius < KRadius) then begin
16         KRadius := MinRadius;
17         for var I: Integer := 1 to CenterCount do
18           KCenters[I] := Centers[I];
19       end;
20     end
21     else
22     ExactFindCenters(CenterCount, K+1, Node+1);
23     //The method calls itself recursively
24   end;
25 end;
```

(a)



(b)

Figure 2. The ExactFindCenters recursion procedure shown in (a) source code and (b) flowchart

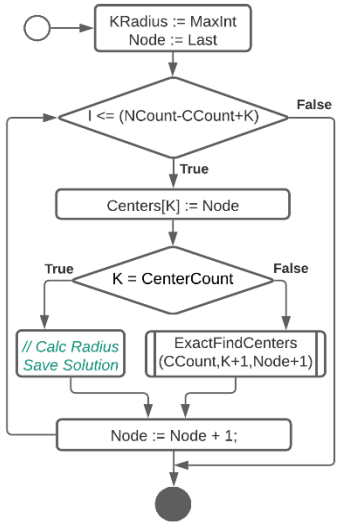
Before starting the ExactFindCenters procedure, the shortest path lengths between each pair of nodes in the graph are calculated. This step is similar to the FindGraphCenter procedure which is used to search only one center in a graph. The idea of searching for multiple centers in a graph (in this case k) is the same as searching for one center. This idea is implemented in the ExactFindCenters procedure (lines 03-20). The difference in searching for more centers is that for each node the greatest distance is minimized not to

exactly one specific center, but to one among several possible ones. Thus, from all the longest distances from a node to each center, the minimum distance is selected. The selection of the analyzed centers is made by generating all possible combinations of k nodes from all available ones. Each subsequent combination is generated by recursively calling the ExactFindCenters procedure (line 22). The number of searched centers (the variable CenterCount), the next element of the subset of the analyzed centers (the variable K + 1) and the next analyzed node (the variable Node + 1) are passed to this procedure as input parameters.

The complexity of this procedure is determined by the number of searched centers (set by the value of the CenterCount parameter). This forms the number of all possible combinations of NodeCount elements of class k-th. For each generated combination, two nested loops are performed (lines 8-10) to minimize the longest distance from a node to some of the centers. In the analysis of the computational complexity, the generation of the Floyd matrix must be taken into account, which is done with a cubic complexity determined by the number of nodes in the graph, i.e., $NodeCount^3$. Figure 3(a) shows the source code and Figure 3(b) shows the flowchart of the ApproximateFindCenters procedure. This code implements an approximate algorithm for finding k-centers in a graph.

```
01 procedure ApproximateFindCenters(CenterCount:Integer);
02 begin
03   Centers[1] := 1; UsedNodes[1] := True;
04   var MinRadius: Integer := 0;
05   for var Index: Integer := 2 to CenterCount do begin
06     var MaxDist: Integer := 0;
07     var MaxIndex: Integer := 0;
08     for var FRow: Integer := 1 to NodeCount do begin
09       if (not UsedNodes[FRow]) then begin
10         var MinDist: Integer := MaxInt;
11         for var FCol: Integer := 1 to Index-1 do
12          ifFloydMatrix[FRow][Centers[FCol]]<MinDist then
13           MinDist:=FloydMatrix[FRow][Centers[FCol]];
14         if (MinDist > MaxDist) then begin
15          MaxDist := MinDist; MaxIndex := FRow; end;
16       end; //end of the if condition
17     end; //end of the for loop
18     Centers[Index] := MaxIndex;
19     MinRadius := MaxDist;
20     UsedNodes[MaxIndex] := True; end;
21 end;
```

(a)



(b)

Figure 3. The ApproximateFindCenters procedure shown in (a) source code and (b) flowchart

When performing this algorithm, one node (the first center of the required k) is initially randomly selected. In this case, this is a node with index 1, which is noted in the arrays Centers and UsedNodes (line 3). Each subsequent center is selected so that it is furthest from the already selected centers. This is done by traversing the nodes in the graph that have not yet been marked as used (line 9). For each of these nodes, the distance from it to the nearest center is checked in the Floyd matrix (among the selected so far, (lines 11-13). Finally, among all the shortest distances the longest one is stored (lines 14-15). Thus, the next searched center in the graph is located. The computational complexity of this approximate algorithm is quadratic and is determined by the number of nodes in the graph and the number of centers [8].

3. RESULTS AND DISCUSSION

The purpose of the experiments is to study the behavior of the algorithms presented in section 2. These algorithms were tested on pre-generated graphs (randomly). The aim is to determine the influence of the number of nodes (respectively the number of edges in a graph) on the time for finding the exact and approximate number of centers. It is necessary to research experimentally for which graphs (in terms of a number of nodes and edges) the exact algorithm can be used to find the optimal solution. The optimal solution is the minimum radius of a graph. In addition, a comparative analysis between the implemented algorithms in terms of solutions found and the execution time needs to be made as well.

3.1. Development of an application for conducting the experiments

A specialized application (named k-center analyzer) was developed for the experiments in this study. An example session of working with the k-center analyzer software is shown in Figure 4. The k-center

analyzer offers a wide range of functions that were used in conducting experiments. It provides the ability to work with graph-type data structures, as well as the implementation of algorithms for finding centers in graphs. The main functions are: creation and editing of graph type structures; maintaining interactive lists of nodes, vertices and edges; maintaining the adjacency and the Floyd matrix; implementation of an exact and an approximate algorithm for finding k-centers in a graph and visualization of the results.

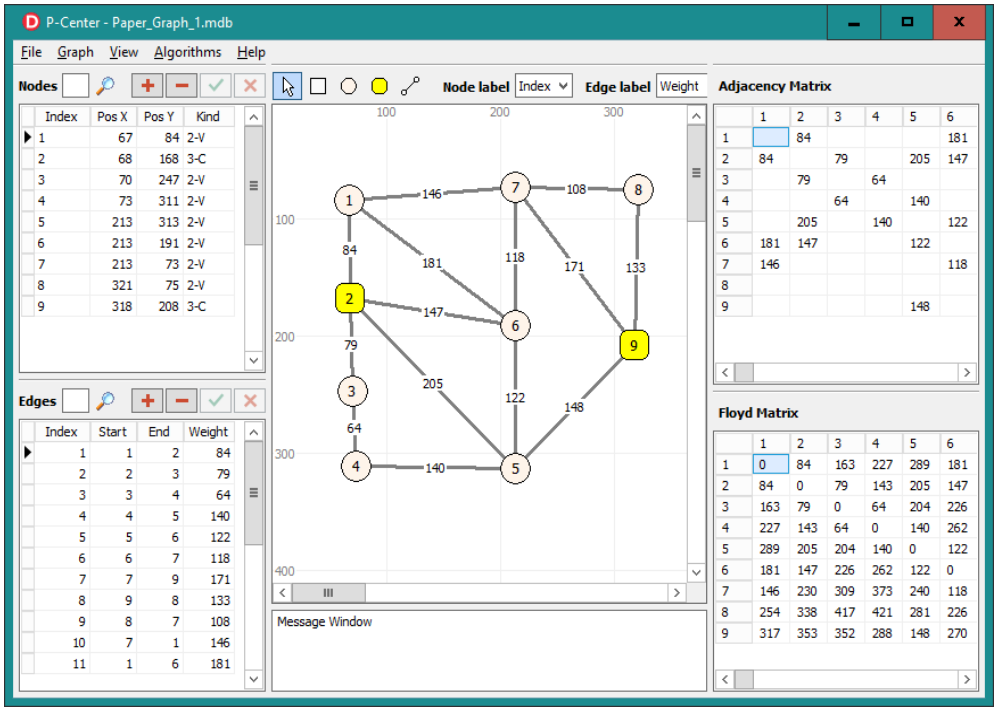


Figure 4. Example session of working with the k-center analyzer software

The main window of the k-center analyzer application is divided into six sections node (vertex) list, edge list, adjacency matrix, floyd matrix, graph designer, and message window. Using draw grids the application visualizes the contents of the dynamic arrays that contain information about the nodes (vertices) and edges in the graph. The adjacency matrix and the floyd matrix are also represented with draw grids. The user can use the control buttons to select the application mode, for example, to add or remove a vertex, node, center or edge, to search by index in the list of vertices or edges and many others. The graph designer tool provides the ability to create and modify a graph by adding and arranging the elements of the graph-vertices, nodes and edges. The message window displays specific information after the execution of certain functions by the application for example, the time to generate the floyd matrix, the execution time of an algorithm, and the result obtained after the execution of this algorithm.

3.2. Experimental results

The results were generated by the k-center analyzer application. The k-center analyzer was run on a computer with 64-bit Win 10 Pro OS with processor: AMD Ryzen 3 Pro 4350 G 3.8 GHz, RAM 16 GB (DDR4-2133), SSD 480 GB. For the experiments 14 graphs were generated randomly, respectively with 20, 30,..., and 150 nodes. Table 1 shows detailed information for those graphs.

The columns "Graph (G)", "Nodes (N)", "Edges (E)", "Vertices (V)", and "Arcs (A)" in Table 1 show, respectively, the name of the graph, the number of nodes, the number of edges, the number of vertices and the number of arcs. The value E/N shows the ratio between the number of edges and the number of nodes. It is noticed that when the number of nodes increases, this ratio also increases. The value A/V shows the ratio between the number of arcs and the number of vertices. It is observed that as the number of vertices increases, this ratio also increases. The values N/V show the ratio between the number of nodes and the number of vertices (in percentages). For each graph, this ratio shows what percentage the nodes relative to the vertices is. It can be seen that when the number of nodes increases, the number of vertices decreases. A similar trend is observed in the ratio between the edges and the arcs ("Ratio (E/A)" column).

Table 2 shows the execution times of the exact algorithm for all 14 analyzed graphs. For each graph, 4 tests were performed, one for each of the four different cases: 2-center, 3-center, 4-center and 5-center. The purpose of this analysis is to determine experimentally how the number of nodes in each graph influences on the execution time of the exact algorithm. The chart in Figure 5 shows the effect of increasing the number of nodes (the x-axis) on the execution time of the exact algorithm (the y-axis in milliseconds).

Table 1. Detailed information about the analyzed graphs

Graph (G)	Nodes (N)	Edges (E)	Vertices (V)	Arcs (A)	Ratio (E/N)	Ratio (A/V)	Ratio (N/V)	Ratio (E/A)
G1	20	46	64	202	2.30	3.16	31.25%	22.77%
G2	30	123	111	603	4.10	5.43	27.03%	20.40%
G3	40	174	164	922	4.35	5.62	24.39%	18.87%
G4	50	312	225	1 778	6.24	7.90	22.22%	17.55%
G5	60	394	294	2 403	6.57	8.17	20.41%	16.40%
G6	70	571	378	3 769	8.16	9.97	18.52%	15.15%
G7	80	794	464	5 637	9.93	12.15	17.24%	14.09%
G8	90	1 023	558	7 570	11.37	13.57	16.13%	13.51%
G9	100	1 292	670	10 207	12.92	15.23	14.93%	12.66%
G10	110	1 473	759	12 079	13.39	15.91	14.49%	12.19%
G11	120	1 765	852	14 650	14.71	17.19	14.08%	12.05%
G12	130	2 079	949	17 672	15.99	18.62	13.70%	11.76%
G13	140	2 432	1 064	21 402	17.37	20.11	13.16%	11.36%
G14	150	2 917	1 155	25 961	19.45	22.48	12.99%	11.24%

Table 2. Execution time of the exact algorithm for the analyzed graphs

Graph Title	2-Centers		3-Centers		4-Centers		5-Centers	
	time (ms)	(h, min, s)	time (ms)	(h, min, s)	time (ms)	(h, min, s)	time (ms)	(h, min, s)
G1	0	< 0.01 s	16	0.02 s	31	0.03 s	110	0.11 s
G2	0	< 0.01 s	31	0.03 s	219	0.22 s	1 453	1.45 s
G3	0	< 0.01 s	93	0.09 s	1 015	1.02 s	9 515	9.51 s
G4	0	< 0.01 s	219	0.22 s	3 265	3.27 s	38 156	38.16 s
G5	15	0.02 s	454	0.45 s	8 641	8.64 s	111 016	1 min, 51 s
G6	31	0.03 s	844	0.84 s	18 875	18.88 s	282 172	4 min, 42 s
G7	38	0.04 s	1 437	1.44 s	38 422	38.42 s	642 578	10 min, 43 s
G8	47	0.05 s	2 360	2.36 s	65 922	1 min, 6 s	1 316 266	21 min, 56 s
G9	78	0.08 s	3 641	3.64 s	113 594	1 min, 54 s	2 464 563	41 min, 05 s
G10	94	0.09 s	5 187	5.19 s	188 766	3 min, 9 s	4 386 844	1 h, 13 min
G11	125	0.13 s	8 344	8.34 s	309 750	5 min, 10 s	7 894 859	2 h, 12 min
G12	157	0.16 s	9 390	9.39 s	385 234	6 min, 25 s	11 642 828	3 h, 14 min
G13	203	0.20 s	12 609	12.61 s	555 719	9 min, 16 s	18 132 156	5 h, 02 min
G14	267	0.27 s	18 735	18.74 s	985 346	16 min, 25 s	31 164 652	8 h, 39 min

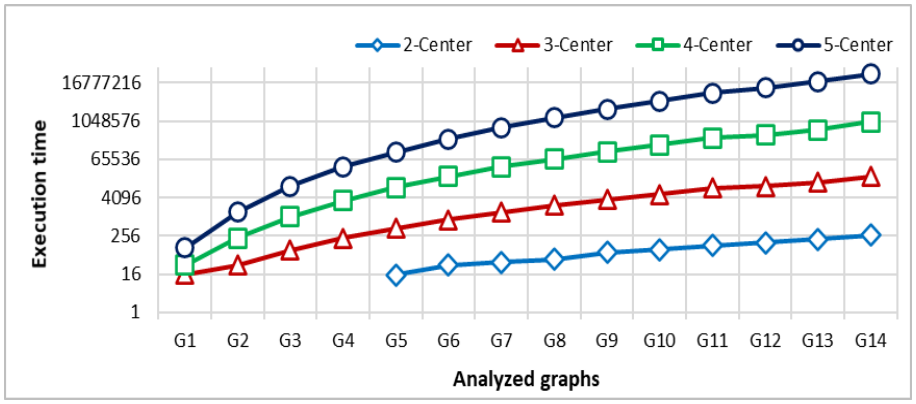


Figure 5. Influence of increasing the number of nodes (x-axis) in each graph on the execution time (y-axis in logarithmic scale with base 2) of the exact algorithm

From the values in Table 2 and the chart of Figure 5 it can be seen that with a linear increase the number of nodes in a graph when searching for 5 centers, the time to execute the exact algorithm increases

exponentially. The aim of the second experiment is to compare and analyze the results of the two algorithms (exact and approximate). The execution time of the approximate algorithm is relatively short (for all studied graphs) and therefore will not be discussed. For each of the studied graphs and number of centers, the ratio between the solution found by the approximate algorithm (A) and the exact algorithm (E) is calculated. This ratio is presented in Table 3 in the columns named "A/E" (Approximate/Exact ratio). These values show the deviation of the found solution from the approximate algorithm to the found solution from the exact one (for each of the analyzed graphs: G1-G14).

The values in the "A/E" columns in Table 3 should be interpreted as follows: a value of 1.00 means that the solution found by the approximate algorithm coincides with the solution found by the exact algorithm. A value of 2.00 means that the solution found by the approximate algorithm is exactly 2 times worse than the optimal solution found by the exact algorithm. The results show that all values in the columns "A/E" are in the range 1.01 and 1.90. This means that when the number of searched centers increases linearly, the radiuses of the graphs decrease linearly.

Table 3. Summarized results after the execution of the exact algorithm and the approximate algorithm.

Graph Title	2-Centers			3-Centers			4-Centers			5-Centers		
	Exact	Approx.	A/E	Exact	Approx.	A/E	Exact	Approx.	A/E	Exact	Approx.	A/E
G1	412	492	1.19	361	363	1.01	199	347	1.74	170	250	1.47
G2	296	446	1.51	283	420	1.48	219	244	1.11	189	224	1.19
G3	414	543	1.31	339	460	1.36	248	332	1.34	221	318	1.44
G4	349	417	1.19	279	384	1.38	210	347	1.65	205	301	1.47
G5	381	632	1.66	308	518	1.68	243	384	1.58	207	307	1.48
G6	363	574	1.58	302	536	1.77	242	388	1.60	207	309	1.49
G7	355	644	1.81	302	488	1.62	229	367	1.60	207	294	1.42
G8	362	577	1.59	299	392	1.31	242	370	1.53	198	283	1.43
G9	368	604	1.64	321	418	1.30	234	339	1.45	214	297	1.39
G10	341	594	1.74	295	466	1.58	225	355	1.58	200	305	1.53
G11	366	611	1.67	314	598	1.90	229	415	1.81	202	293	1.45
G12	364	600	1.65	318	495	1.56	231	364	1.58	212	304	1.43
G13	361	579	1.60	322	553	1.72	222	377	1.70	200	282	1.41
G14	356	583	1.64	317	516	1.63	227	372	1.64	196	276	1.41

4. CONCLUSION

In this paper, a research of the k-center problem hase been presented. Various methods and algorithms to its solution have been analyzed. The implementations of two algorithms (one exact and one approximate) have been also described. The definitions of the dynamic data structures one-dimensional arrays and two-dimensional arrays (matrices) have been shown as well. The codes of the algorithm procedures have been implemented and analyzed. When considering the execution time of the algorithms, the multitasking mode of operation of the operating system ihas been taken into account. The methodology for the experiments, the aim of the research, and the conditions for implementing the experiments have been described. For the experiments, fourteen graphs were generated randomly. The ratios between the number of edges and the number of nodes, the number of arcs and the number of vertices, the number of nodes and the number of vertices, and the ratio between the edges and the arcs were calculated. A software was developed for this research. It implemented the described algorithms. Its main functions have been presented as well. All the results in this research were generated by this software. From the results it can be concluded that the approximate algorithm finds solutions that are not worse than 2 times optimal. These solutions in some cases are very close to the optimal solutions, but this is true only for graphs with a smaller number of nodes, respectively edges, such as for graphs G1-G4. As the number of nodes in the graph increases (respectively the number of edges), the approximate solutions found deviate from the optimal ones, but remain acceptable, for example in graphs G5-G9. These results give reason to conclude that for graphs with a smaller number of vertices (respectively edges) the approximate algorithm finds comparable solutions with those of the exact algorithm, even for a larger number of centers. In addition, the execution time of the approximate algorithm is significantly less than the execution time of the exact algorithm.

REFERENCES

[1] V. E. Alekseev, R. Boliac, D. V Korobitsyn, and V. V Lozin, "NP-hard graph problems and boundary classes of graphs," *Theoretical Computer Science*, vol. 389, no. 1–2, pp. 219–236, Dec. 2007, doi: 10.1016/j.tcs.2007.09.013.

[2] S. V Kurapov, M. V Davidovsky, and A. V Tolok, "A modified algorithm for planarity testing and constructing the topological drawing of a graph. The thread method," *Scientific Visualization*, vol. 10, no. 4, pp. 53–74, Oct. 2018, doi: 10.26583/sv.10.4.05.

[3] B. L. Natarajan, "Computation of chromatic numbers for new class of graphs and its applications," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 8, pp. 396–400, 2019.
[4] R. Hidayat, I. Tri Riyadi Yanto, A. Azhar Ramli, M. Farhan Md. Fudzee, and A. Saleh Ahmar, "Generalized normalized euclidean distance based fuzzy soft set similarity for data classification," *Computer Systems Science and Engineering*, vol. 38, no. 1, pp. 119–130, 2021, doi: 10.32604/csse.2021.015628.
[5] A. K. Abdulsahib and S. S. Kamaruddin, "Graph based text representation for document clustering," *Journal of Theoretical and Applied Information Technology*, vol. 76, no. 1, pp. 1–13, 2015.
[6] S. Sabeen, R. Arunadevi, B. Kanisha, and R. Kesavan, "Mining of sequential patterns using directed graphs," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 11, pp. 4002–4007, Sep. 2019, doi: 10.35940/ijitee.K2242.0981119.
[7] T. Kosar, S. Gaberc, J. C. Carver, and M. Mernik, "Program comprehension of domain-specific and general-purpose languages: replication of a family of experiments using integrated development environments," *Empirical Software Engineering*, vol. 23, no. 5, pp. 2734–2763, Oct. 2018, doi: 10.1007/s10664-017-9593-2.
[8] R. Rana and D. Garg, "Heuristic approaches for k-center problem," in *2009 IEEE International Advance Computing Conference*, Mar. 2009, pp. 332–335, doi: 10.1109/IADCC.2009.4809031.
[9] D. Z. Chen, J. Li, and H. Wang, "Efficient algorithms for the one-dimensional k-center problem," *Theoretical Computer Science*, vol. 592, pp. 135–142, Aug. 2015, doi: 10.1016/j.tcs.2015.05.028.
[10] D. Matic, J. Kratica, and Z. Maksimovic, "Solving the minimum edge-dilation k-center problem by genetic algorithms," *Computers & Industrial Engineering*, vol. 113, pp. 282–293, Nov. 2017, doi: 10.1016/j.cie.2017.09.029.
[11] R. Ge, M. Ester, B. J. Gao, Z. Hu, B. Bhattacharya, and B. Ben-Moshe, "Joint cluster analysis of attribute data and relationship data," *ACM Transactions on Knowledge Discovery from Data*, vol. 2, no. 2, pp. 1–35, Jul. 2008, doi: 10.1145/1376815.1376816.
[12] H. Du and Y. Xu, "An approximation algorithm for k-center problem on a convex polygon," *Journal of Combinatorial Optimization*, vol. 27, no. 3, pp. 504–518, Apr. 2014, doi: 10.1007/s10878-012-9532-5.
[13] N. T. An, N. M. Nam, and X. Qin, "Solving k-center problems involving sets based on optimization techniques," *Journal of Global Optimization*, vol. 76, no. 1, pp. 189–209, Jan. 2020, doi: 10.1007/s10898-019-00834-6.
[14] D. Chakrabarty, P. Goyal, and R. Krishnaswamy, "The Non-Uniform k -Center Problem," *ACM Transactions on Algorithms*, vol. 16, no. 4, pp. 1–19, Sep. 2020, doi: 10.1145/3392720.
[15] J. A. Cornejo Acosta, J. García Díaz, R. Menchaca-Méndez, and R. Menchaca-Méndez, "Solving the capacitated vertex k-center problem through the minimum capacitated dominating set problem," *Mathematics*, vol. 8, no. 9, Sep. 2020, doi: 10.3390/math8091551.
[16] M. Basappa, R. K. Jallu, and G. K. Das, "Constrained k-center problem on a convex polygon," *International Journal of Foundations of Computer Science*, vol. 31, no. 2, pp. 275–291, Feb. 2020, doi: 10.1142/S0129054120500070.
[17] A. E. Feldmann, "Fixed-parameter approximations for k-center problems in low highway dimension graphs," *Algorithmica*, vol. 81, no. 3, pp. 1031–1052, Mar. 2019, doi: 10.1007/s00453-018-0455-0.
[18] A. E. Feldmann and D. Marx, "The parameterized hardness of the k-center problem in transportation networks," *Algorithmica*, vol. 82, no. 7, pp. 1989–2005, Jul. 2020, doi: 10.1007/s00453-020-00683-w.
[19] H. Wang and J. Zhang, "An O(n log n)-time algorithm for the k-center problem in trees," *SIAM Journal on Computing*, vol. 50, no. 2, pp. 602–635, Jan. 2021, doi: 10.1137/18M1196522.
[20] J. Garcia-DIaz, R. Menchaca-Mendez, R. Menchaca-Mendez, S. Pomares Hernandez, J. C. Perez-Sansalvador, and N. Lakouari, "Approximation algorithms for the vertex k-center problem: survey and experimental evaluation," *IEEE Access*, vol. 7, pp. 109228–109245, 2019, doi: 10.1109/ACCESS.2019.2933875.
[21] H. Du, Y. Xu, and B. Zhu, "An incremental version of the k-center problem on boundary of a convex polygon," *Journal of Combinatorial Optimization*, vol. 30, no. 4, pp. 1219–1227, Nov. 2015, doi: 10.1007/s10878-015-9933-3.
[22] J. Garcia, R. Menchaca, R. Menchaca, and R. Quintero, "A structure-driven randomized algorithm for the k-center problem," *IEEE Latin America Transactions*, vol. 13, no. 3, pp. 746–752, Mar. 2015, doi: 10.1109/TLA.2015.7069100.
[23] S. Chechik and D. Peleg, "The fault-tolerant capacitated k-center problem," *Theoretical Computer Science*, vol. 566, pp. 12–25, Feb. 2015, doi: 10.1016/j.tcs.2014.11.017.
[24] H. Wang and J. Zhang, "Line-constrained k-median, k-means, and k-center problems in the plane," *International Journal of Computational Geometry and Applications*, vol. 26, pp. 185–210, Sep. 2016, doi: 10.1142/S0218195916600049.
[25] V. Kralev, "Different applications of the genetic mutation operator for symetric travelling salesman problem," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 8, no. 3, Jun. 2018, doi: 10.18517/ijaseit.8.3.4867.
[26] D. Liang, L. Mei, J. Willson, and W. Wang, "A simple greedy approximation algorithm for the minimum connected k-Center problem," *Journal of Combinatorial Optimization*, vol. 31, no. 4, pp. 1417–1429, May 2016, doi: 10.1007/s10878-015-9831-8.
[27] V. Kralev and R. Kraleva, "Methods for software visualization of large graph data structures," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 10, no. 1, Feb. 2020, doi: 10.18517/ijaseit.10.1.10739.
[28] V. S. Kralev and R. S. Kraleva, "Visual analysis of actions performed with big graphs," *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, no. 1, pp. 2740–2744, Nov. 2019, doi: 10.35940/ijitee.A4978.119119.
[29] V. Kralev, "An analysis of a recursive and an iterative algorithm for generating permutations modified for travelling salesman problem," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 7, no. 5, Oct. 2017, doi: 10.18517/ijaseit.7.5.3173.
[30] J. Garcia, R. Menchaca, J. Sanchez, and R. Menchaca, "Local search algorithms for the vertex k-center problem," *IEEE Latin America Transactions*, vol. 16, no. 6, pp. 1765–1771, Jun. 2018, doi: 10.1109/TLA.2018.8444397.