UDC 004.8

Maksym Bondarchuk,
Heorhii Loutskii, Artem Volokyta.

GENETIC ALGORITHM APPLIANCE FOR TRAVELING SALESMAN PROBLEM FOR GRAPHS WITH DE BRUIJN TOPOLOGY

Article reviews appropriateness for genetic algorithms to be used for traveling salesman problem for graphs with De Bruijn topologies. For this reason, article reviews requirements for graph and verifies whether De Bruijn topology satisfies requirements. Article also presents new crossover function and compares its quality with two existing functions.

Keywords: De Bruijn topology, genetic algorithm, traveling salesman problem.

Fig.: 11. Tabl.: 3. Bibl.: 7.

Actuality of theme. Traveling salesman problem (TCP) was important from ancient ages since first humans settled. Brute force solution becomes very complex for computing even with 20 cities. Genetic algorithms took their chance to solve this problem in a way to propose more or less good solution in short time with less computing. Author was unable to find appliance of genetic algorithm of such kind for De Bruijn graph built using excess coding.

Target setting. Absence of genetic algorithms for traveling salesman problem for graphs with De Bruijn topology and unclearness of requirements for graphs for genetic algorithm.

Actual scientific studies and issues analysis. [1] reviews cycle crossover approach claiming that it has higher quality over order crossover. We will compare and test these two approaches. Some studies try to use knowledge-augmented crossover operation [2]. Some articles implement more advanced evolutionary algorithms such as particle swarm optimization for TCP [3]. Some articles try to replace static parameters of algorithm with dynamic ones and smarter selection of initial population instead of random distribution [4]. Almost all reviewed articles do not describe the graph topology that was used. [5] makes a wide overview of De Bruijn topology with excess coding. We will use information from this article to investigate topology's appliance for genetic algorithm.

Uninvestigated parts of general matters defining. This article focuses on defining requirements for graph topology for genetic algorithm to be able to find an optimal path for traveling salesman problem. Genetic algorithms for traveling salesman problem is a well-researched area, but test results and sets raise some questions. De Bruijn topology for traveling salesman problem is not covered with latest studies.

Problem formulation. It is unclear at this moment, which graph topology is acceptable for TCP and which is not. With that said, the main goal of this article is to set up some requirements for graph to be able to be processed by genetic algorithm.

The statement of basic materials. Genetic algorithms are inspired by evolutionary development of breeds in nature. The main idea is to imitate natural selection and mutations and apply them for solution of computer science problems. Each genetic algorithm operates with population of members called chromosomes. In general, genetic algorithm consists of 3 main terms: fitness, crossover and mutation.

Fitness function or just fitness represents the ability of chromosome to survive in natural environment and survive selection to next generation. In this article we will use smaller fitness values for chromosomes of better quality because we are minimizing path between graph vertices (cities in TCP). In other words, the smaller fitness is, the better. Fitness value will be calculated as a sum of edges' weights of path. So, each chromosome contains path that it represents and fitness value. Each path contains all vertices of the graph without duplicates.

Crossover is an algorithm of producing offspring from two parent chromosomes by exchanging their genes. Crossover function attracts the highest attention in studies. Crossover phase requires selection phase to select fittest parents for mating. We will use tournament selection [**Ошибка! Источник ссылки не найден.**].

Mutation phase is inspired by mutations in different breeds (e.g. dogs) and commonly represents minor change in some portion of chromosomes. The idea behind mutation is to save algorithm from premature convergence.

TSP uses permutation kind of genetic algorithm where chromosome is a permutation of finite set of items.

First analyzed crossover is order function (OX). Both parents are split at the same random point called crossover point. Then the part rightward from first parent chromosome is put at the same positions in offspring chromosome. Then we go through second parent chromosomes from left and put unused in offspring chromosomes at empty places from left. Fig. 1 shows parents' chromosomes before crossover. Let crossover point be at 6th place.

7	5	2	1	4	3	8	6

1	2	4	8	6	5	7	3

Fig. 1. Parents' chromosomes for OX

Fig. 2 shows offspring chromosome.

1	2	4	5	7	3	8	6

Fig. 2. Offspring's chromosome for OX

Order crossover chromosome is very popular and easy in implementation.

Second analyzed crossover is cycle function (CX). From first sight cycle crossover looks smarter than order one, but we will test and compare them. The idea of cycle crossover is to proportionally pick genes from both parents and produce offspring of higher quality. In this function we move from left side while picking genes from first parent and then swapping them. If picked gene is already present in offspring's chromosome, we try to pick a new one from second parent. If we fail with second parent too, then we start moving from left side again and peek first unused gene from both parents. Fig. 3 shows the result of parents crossing from Fig. 1.

7	2	4	1	6	3	8	5

Fig. 3. Offspring's chromosome for CX. Case 1

1	5	3	2	7	6	8	4

8	2	4	1	6	5	7	3

Fig. 4. Parents' chromosomes for CX

Fig. 5 covers case where gene cannot be selected from any parent (at 4^{th} place).

1	2	3	8	7	5	4	6

Fig. 5. Offspring's chromosome for CX. Case 2

Third analyzed crossover is random function (RX). This is a new crossover developed by author. The idea was to randomize crossover as much as possible. We go through both parents at the same time and for each gene select random from any parent (even if it duplicates gene in offspring) – first stage. After child has all genes (most likely with duplicates) we check duplicated genes in offspring and unused genes from both parents – second stage. Then we swap sets of duplicated (last entrance of duplicate) and unused genes – third stage. Suppose that first stage produced genes as on Fig. 6 for parents from Fig. 4.

1	2	4	2	7	5	8	4

Fig. 6. First stage genes for RX

Figs. 7 and 8 show duplicated and unused genes.

2	4

Fig. 7. Duplicated genes from first stage

3	6

Fig. 8. Unused genes from parents

After genes from Figs. 7 and 8 are swapped we receive an offspring's chromosome.

1	3	4	2	7	5	8	6

Fig. 9. Offspring's chromosome for RX

As mutation we just swap two pseudo-random vertices in 20% of population's chromosomes. Figs. 10 and 11 show mutation on example.

7	5	2	1	4	3	8	6

Fig. 10. Chromosome before mutation

7	5	3	1	4	2	8	6

Fig. 11. Chromosome after mutation

De Bruijn graphs are generated by overlapping of vertices numbers. Each vertex is usually coded with binary code and to generate edges to other nodes we shift left and right its value and put all available characters instead of released character. In case of binary code, vertex 10 will be connected with 00 (shift left and put 0), 01 (shift left and put 1 or shift right and put 0) and 11 (shift right and put 1). We will analyze three cases of excess coding:

- with 3 possible values: -1 (T), 0 and 1 called T system;
- with 5 possible values: -2 (Z), -1 (T), 0, 1, 2 called Z system;
- with 7 possible values: -3 (E), -2 (Z), -1 (T), 0, 1, 2, 3 called E system.

For all three cases each vertex will have higher power (number of edges). For T system vertex T01 will be connected with 01T (shift left and put T), 010 (shift left and put 0), 011 (shift left and put 1), TT0 (shift right and put T), 0T0 (shift right and put 0) and 1T0 (shift right and put 1).

In such topology some vertices may have the same digital system value (e.g. T0T = TT1 = 5). We organize such vertices in a cluster and connect them with each other using edges of minimal weight (value is 1).

Tests and analysis. To test genetic algorithm, the program was written using C# language [**Ошибка! Источник ссылки не найден.**]. The goal of a program is to test three analyzed crossover functions for different graphs with different degrees and to figure out dependency between graph type and success rate. Program was run for graphs with vertices number from 5 to 20 and degree from 1 to number of vertices. For each such combination, tests with chromosomes number from 2 to 10 were run. For each 'vertices number – graph degree' pair 10 different graphs were generated and 12 tests were run for each crossover function, then first and last tests were excluded and average result was collected. With such approach, program generated 1800 results that will require 30 pages to present them in this article, so only some results are presented in Appendix A with graphs' combinations for which algorithms starts to be able to find an optimal path (before such combination for given vertices number algorithm is either unable to find a path for all 3 crossovers or finds a path for all of them).

Table 1 contains approximate degree for which algorithm was able to find an optimal path (the best or close to the best) for more than 50% of tests. We see that for each ‘vertices number – graph degree’ pair the ratio between graph degree and number of vertices is approximately the same and equals 0.57. We also see that OX found optimal paths in more cases compared to other crossovers.

This test gives us a conclusion that in order for genetic algorithm with order, cycle or random crossover to be able to find an optimal path, the ratio between graph degree and number of vertices must be at least 0.52.

Table 1

Algorithm raw successful results

V	Dgr OX	Rt OX	Dgr CX	Rt CX	Dgr RX	Rt RX	Dgr Avg	Rt Avg
5	3	0,60	4	0,80	4	0,80	4	0,80
6	3	0,50	4	0,67	4	0,67	4	0,67
7	4	0,57	4	0,57	4	0,57	4	0,57
8	4	0,50	5	0,63	5	0,63	5	0,63
9	4	0,44	5	0,56	5	0,56	5	0,56
10	5	0,50	6	0,60	6	0,60	6	0,60
11	5	0,45	6	0,55	6	0,55	6	0,55
12	6	0,50	7	0,58	6	0,50	6	0,50
13	7	0,54	7	0,54	7	0,54	7	0,54
14	7	0,50	7	0,50	7	0,50	7	0,50
15	8	0,53	8	0,53	8	0,53	8	0,53
16	8	0,50	8	0,50	8	0,50	8	0,50
17	9	0,53	9	0,53	9	0,53	9	0,53
18	10	0,56	10	0,56	9	0,50	10	0,56
19	11	0,58	10	0,53	10	0,53	11	0,58
20	11	0,55	11	0,55	11	0,55	11	0,55
		0,52		0,57		0,57		0,57

where *V* is a number of vertices, *Dgr* – degree of a graph, *Rt* – graph degree to number of vertices ration and *Avg* is an average value for three crossover functions.

Table 2 presents the same ratio for De Bruijn graphs for vertices’ sets with bitness (number of characters from alphabet to describe node) 2, 3 and 4 for T, Z and E systems. All results are generated with the same program.

Table 2

De Bruijn graphs’ degree and degree to vertices ratio

Bitness	System	Vertices	Degree	Ratio
2	T	9	4,67	0,52
3	T	27	6,44	0,24
4	T	81	8,02	0,10
2	Z	25	9,36	0,37
3	Z	125	13,71	0,11

Ended Table 2

Bitness	System	Vertices	Degree	Ratio
4	Z	625	22,16	0,04
2	E	49	14,12	0,29
3	E	343	22,64	0,07
4	E	2401	48,53	0,02

In this table we see that only one graph meets the conditions of ratio value (and algorithm finds an optimal path). If we continue to increase bitness or system (add extra values: 9, 11 characters and so on), the ratio will be decreasing.

Conclusions. Results show us that order crossover has higher quality then cycle or random. Results also show that graph degree to number of vertices ratio should be at least 0.52 for genetic algorithm to be able to find optimal path in graph for TSP. We also see that De Bruijn topology does not meet the requirements for such algorithm. Proposed random crossover function is not worse than cycle crossover and for some tests even shows better results.

The impact of chromosomes number for 'vertices number – graph degree' pairs in places where algorithm starts to be able to find an optimal path can be the prospect of further study.

Appendix A.

Test result of order, cycle and random crossovers for different graph topologies

V	Dg	Ch	I OX	P OX	F OX	I CX	P CX	F CX	I RX	P RX	F RX	I Av	P Av	F Av
…														
5	2	10	24	321691	No	55	321691	No	58	321691	No	46	321691	No
5	3	2	59	1575	Yes	15	261614	No	15	211617	No	29	158269	No
5	3	3	47	1755	Yes	14	271731	No	13	231762	No	24	168416	No
5	3	4	35	1695	Yes	14	61871	No	15	91821	No	21	51796	No
5	3	5	49	1467	Yes	16	61744	No	17	91720	No	27	51643	No
5	3	6	34	1378	Yes	85	1378	Yes	96	1378	Yes	72	1378	Yes
5	3	7	32	1539	Yes	93	1539	Yes	98	1539	Yes	74	1539	Yes
5	3	8	31	81407	No	71	81407	No	66	81407	No	56	81407	No
5	3	9	30	1157	Yes	88	1157	Yes	93	1157	Yes	70	1157	Yes
5	3	10	34	1583	Yes	72	1583	Yes	62	1583	Yes	56	1583	Yes
5	4	2	40	814	Yes	13	1210	Yes	14	1184	Yes	22	1070	Yes
5	4	3	47	1258	Yes	14	21395	No	14	21428	No	24	14694	No
5	4	4	37	1321	Yes	16	1535	Yes	17	11506	No	23	4787	Yes
5	4	5	43	1091	Yes	17	11350	No	13	11307	No	24	7916	No
5	4	6	44	1322	Yes	97	1323	Yes	98	1322	Yes	80	1323	Yes
…														
8	3	10	452	52474	No	522	52514	No	537	22579	No	503	42522	No
8	4	2	563	2051	Yes	43	262532	No	47	282514	No	217	182366	No
8	4	3	503	1953	Yes	62	172581	No	61	212592	No	208	129042	No
8	4	4	520	2166	Yes	53	352425	No	51	372506	No	208	242366	No
8	4	5	503	1909	Yes	62	182555	No	72	182588	No	212	122351	No
8	4	6	591	2006	Yes	625	2009	Yes	432	2010	Yes	549	2008	Yes

Continuation appendix A.

V	Dg	Ch	I OX	P OX	F OX	I CX	P CX	F CX	I RX	P RX	F RX	I Av	P Av	F Av
8	4	7	577	2239	Yes	547	2254	Yes	527	2289	Yes	550	2261	Yes
8	4	8	501	2258	Yes	586	2335	Yes	532	2300	Yes	540	2298	Yes
8	4	9	555	2016	Yes	633	2100	Yes	576	2082	Yes	588	2066	Yes
8	4	10	542	2064	Yes	625	2087	Yes	523	2105	Yes	563	2085	Yes
8	5	2	492	1840	Yes	52	32492	No	54	62431	No	199	32255	No
8	5	3	538	1679	Yes	46	92274	No	45	72289	No	210	55414	No
8	5	4	507	1597	Yes	69	62286	No	64	82239	No	213	48707	No
8	5	5	499	1857	Yes	65	12476	No	72	82388	No	212	32241	No
8	5	6	508	1910	Yes	550	1937	Yes	599	1928	Yes	552	1925	Yes
8	5	7	578	1868	Yes	591	1921	Yes	530	1898	Yes	566	1896	Yes
8	5	8	595	81785	No	577	81827	No	599	81811	No	590	81808	No
8	5	9	469	1651	Yes	569	1674	Yes	598	1674	Yes	545	1667	Yes
8	5	10	615	1750	Yes	489	1808	Yes	601	1789	Yes	568	1783	Yes
8	6	2	614	1333	Yes	49	11846	No	50	1872	Yes	238	5017	Yes
8	6	3	520	1641	Yes	48	12139	No	65	12120	No	211	8633	No
8	6	4	616	1611	Yes	56	2137	Yes	61	2191	Yes	244	1979	Yes
…														
11	4	10	655	263706	No	600	123834	No	645	63829	No	633	150456	No
11	5	2	687	2878	Yes	100	183499	No	106	253405	No	298	146594	No
11	5	3	732	13087	No	135	283427	No	128	363432	No	331	219982	No
11	5	4	666	2529	Yes	176	233287	No	145	233146	No	329	156321	No
11	5	5	707	2785	Yes	148	353344	No	129	283455	No	328	213194	No
11	5	6	573	203313	No	641	193324	No	601	193297	No	605	196645	No
11	5	7	673	3391	Yes	761	3283	Yes	805	13171	No	746	6615	Yes
11	5	8	665	3019	Yes	615	3043	Yes	590	3030	Yes	623	3031	Yes
11	5	9	676	3263	Yes	700	3215	Yes	763	13266	No	713	6581	Yes
11	5	10	542	33059	No	667	3089	Yes	693	13084	No	634	16411	No
11	6	2	659	2486	Yes	105	43161	No	101	33253	No	288	26300	No
11	6	3	631	2533	Yes	146	113268	No	141	53363	No	306	56388	No
11	6	4	713	2498	Yes	130	73226	No	136	73324	No	326	49682	No
11	6	5	696	2414	Yes	158	53202	No	138	73096	No	330	42904	No
11	6	6	773	2761	Yes	623	2750	Yes	711	2731	Yes	702	2747	Yes
11	6	7	746	2755	Yes	640	2704	Yes	583	2684	Yes	656	2715	Yes
11	6	8	600	2934	Yes	666	2861	Yes	629	2819	Yes	631	2871	Yes
11	6	9	665	2495	Yes	623	2476	Yes	607	2478	Yes	631	2483	Yes
11	6	10	635	2718	Yes	652	2609	Yes	663	2583	Yes	649	2637	Yes
11	7	2	661	2325	Yes	117	2853	Yes	97	12913	No	292	6031	Yes
11	7	3	655	2335	Yes	141	12868	No	122	12922	No	306	9375	Yes
11	7	4	613	2320	Yes	159	22785	No	145	32929	No	306	19345	No
11	7	5	739	2073	Yes	131	2755	Yes	157	2771	Yes	342	2533	Yes
…														
12	5	2	726	43643	No	131	424005	No	136	423977	No	331	297208	No
12	5	3	764	3425	Yes	154	403640	No	151	363557	No	356	256874	No
12	5	4	630	13681	No	181	393907	No	185	373899	No	332	260496	No
12	5	5	706	3169	Yes	192	353580	No	167	333719	No	355	230156	No
12	5	6	670	123961	No	696	73947	No	644	53934	No	670	83947	No
12	5	7	613	53982	No	792	33823	No	596	3824	Yes	667	30543	No
12	5	8	637	123776	No	657	63775	No	692	33775	No	662	73775	No
12	5	9	724	123770	No	694	53839	No	658	63726	No	692	80445	No
12	5	10	616	143887	No	624	83811	No	805	83618	No	681	103772	No
12	6	2	649	3234	Yes	119	103758	No	151	183608	No	306	96866	No
12	6	3	672	3231	Yes	152	183726	No	156	163718	No	327	116892	No
12	6	4	640	3171	Yes	193	103650	No	182	123742	No	338	76855	No
12	6	5	626	2987	Yes	204	43883	No	198	103610	No	342	50160	No

Ended appendix A.

V	Dg	Ch	I OX	P OX	F OX	I CX	P CX	F CX	I RX	P RX	F RX	I Av	P Av	F Av
12	6	6	561	3577	Yes	672	3605	Yes	678	3587	Yes	637	3589	Yes
12	6	7	687	3385	Yes	659	3306	Yes	656	3207	Yes	667	3299	Yes
12	6	8	631	23225	No	577	13183	No	726	3147	Yes	644	13185	No
12	6	9	713	3475	Yes	734	3233	Yes	639	3292	Yes	695	3333	Yes
…														
14	6	3	731	24162	No	206	234175	No	259	294031	No	398	184122	No
14	6	4	735	4450	Yes	298	254587	No	261	254586	No	431	171208	No
14	6	5	715	4107	Yes	246	324329	No	233	304228	No	398	210888	No
14	6	6	629	204852	No	621	104898	No	634	74742	No	628	128164	No
14	6	7	621	124827	No	697	94716	No	717	104604	No	678	108049	No
14	6	8	739	94913	No	633	54616	No	624	64571	No	665	71367	No
14	6	9	595	124660	No	735	44409	No	640	64376	No	656	77815	No
14	6	10	668	134515	No	776	34439	No	745	54392	No	729	74449	No
14	7	2	674	3836	Yes	194	114135	No	184	84102	No	350	67358	No
14	7	3	710	3650	Yes	244	24166	No	231	54101	No	395	27306	No
14	7	4	680	3541	Yes	264	124068	No	274	24119	No	406	50576	No
14	7	5	604	3522	Yes	222	64192	No	310	73987	No	378	47234	No
14	7	6	616	4391	Yes	655	4158	Yes	666	4170	Yes	645	4240	Yes
14	7	7	609	24440	No	658	4227	Yes	779	4217	Yes	682	10961	Yes
14	7	8	702	4345	Yes	669	4131	Yes	666	4107	Yes	679	4194	Yes
14	7	9	728	4123	Yes	768	3907	Yes	670	3812	Yes	722	3947	Yes
14	7	10	528	4169	Yes	746	3844	Yes	691	3743	Yes	655	3918	Yes
14	8	2	638	3295	Yes	203	23657	No	201	33704	No	347	20219	No
14	8	3	673	3495	Yes	267	33873	No	256	13865	Yes	399	17077	No
14	8	4	648	3238	Yes	296	3581	Yes	308	3564	Yes	417	3461	Yes
14	8	5	635	3153	Yes	263	3616	Yes	320	43546	No	406	16772	No
14	8	6	657	3730	Yes	634	3662	Yes	528	3706	Yes	606	3699	Yes
…														
20	9	10	669	457059	No	754	176888	No	808	196835	No	744	276928	No
20	10	2	813	5588	Yes	489	15288	Yes	414	35284	No	572	18720	Yes
20	10	3	711	16289	Yes	619	65368	No	589	15391	Yes	640	32349	No
20	10	4	706	5992	Yes	631	35502	No	577	35504	No	638	25666	No
20	10	5	741	5736	Yes	659	5460	Yes	702	25483	No	700	12226	Yes
20	10	6	649	326750	No	670	86396	No	809	46515	No	709	153220	No
20	10	7	710	267052	No	769	66631	No	663	26859	No	714	120181	No
20	10	8	749	256683	No	633	96378	No	663	76377	No	681	143146	No
20	10	9	647	236734	No	711	26487	No	735	56304	No	697	106508	No
20	10	10	608	166666	No	738	76158	No	778	36344	No	708	93056	No
20	11	2	660	5213	Yes	488	4517	Yes	565	4651	Yes	571	4793	Yes
20	11	3	771	5554	Yes	632	4843	Yes	619	4855	Yes	674	5084	Yes
20	11	4	710	5390	Yes	648	24803	No	736	4872	Yes	698	11688	Yes
20	11	5	676	5284	Yes	764	4720	Yes	735	4885	Yes	725	4963	Yes
20	11	6	762	46627	No	758	6325	Yes	683	16006	Yes	734	22986	No
20	11	7	792	36524	No	744	5954	Yes	786	5854	Yes	774	16110	Yes
20	11	8	717	16362	Yes	681	5764	Yes	722	5814	Yes	706	9313	Yes
20	11	9	691	16478	Yes	698	5998	Yes	764	5807	Yes	718	9428	Yes
20	11	10	718	26420	No	744	5868	Yes	867	5821	Yes	776	12703	Yes
20	12	2	651	5007	Yes	445	4593	Yes	524	14407	Yes	540	8002	Yes
…														

where *V* is a vertices number, *Dg* – graph degree, *Ch* – chromosomes number, *I* – last improvement iteration number, *P* – found path’s length, *F* – flag that optimal path was found, *Av* – average value for OX, CX and RX.

References

1. Conforto, S., Hussain, A., Muhammad, Y.S., Nauman Sa., Hussain, I., Mohamd Shoukry & A., Gani, S. (2017). “Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator”, Computational Intelligence and Neuroscience, Hindawi. DOI 10.1155/2017/7430125.

2. Pál, K.F. (1993). “Genetic algorithms for the traveling salesman problem based on a heuristic crossover operation”. Biological Cybernetics. Vol. 69, pp. 539–546. DOI 10.1007/BF01185425.

3. Liao, Y.-F., Yau, D.-H. & Chen, C.-L. (2012). “Evolutionary algorithm to traveling salesman problems”. Computers & Mathematics with Applications. Vol. 64, Iss. 5, pp. 788-797. DOI 10.1016/j.camwa.2011.12.018.

4. Kaabi J. & Harrath Y. (2019). “Permutation rules and genetic algorithm to solve the traveling salesman problem”. Arab Journal of Basic and Applied Sciences. Vol. 26, Iss. 1. DOI 10.1080/25765299.2019.1615172.

5. Loutskii H., Volokyta A., Rehida P. & Goncharenko O. (2019). “Using excess code to design fault-tolerant topologies”. Technical sciences and technologies. National University "Chernihiv Polytechnic". Vol. 1 (15). DOI 10.25140/2411-5363-2019-1(15)-134-144.

6. Li J. (2010). “A Review of Tournament Selection in Genetic Programming”. Advances in Computation and Intelligence: 5th International Symposium, ISICA 2010, China, Wuhan. pp.181-192. DOI 10.1007/978-3-642-16493-4_19.

7. Bondarchuk M. Genetic De Bruijn Travelling Salesman Problem. GitHub repository. [Electronic resource]. – Available at: https://github.com/MaxBondarchuk/ Genetic DeBruijnTravellingSalesmanProblem. – Active link: 06.06.2020.

AUTHORS

Maksym Bondarchuk – PhD student, Specialized Computer Systems Department, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.

E-mail: bondarchuk.m.y@gmail.com

Artem Volokyta (supervisor) – associate professor, Department of Computer Engineering, Igor Sikorsky Kyiv Polytechnic Institute”.

Heorhii Loutskii (supervisor) – professor, Department of Computer Engineering, Igor Sikorsky Kyiv Polytechnic Institute.