Date of publication DEC-31, 2021, date of current version NOV-02, 2021.
www.computingonline.net / computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.20.4.2444

Fast Algorithm for Calculating Transitive Closures of Binary Relations in the Structure of a Network Object

VLADIMIR BATSAMUT, SVIATOSLAV MANZURA, OLEKSANDR KOSIAK, VIACHESLAV GARMASH, DMYTRO KUKHARETS

Research Center, National Academy of the National Guard of Ukraine, 3, Zakhysnykiv Ukrayny, Kharkiv, Ukraine, (e-mail: batsamut@ukr.net, ivanguvv@gmail.com, kosoj12345@gmail.com, 2708garmash@gmail.com, 13dkma@gmail.com), www. nangu.edu.ua

Corresponding author: Vladimir Batsamut (e-mail: batsamut@ukr.net).

This research is supported by the project № 0120U002173 funded by the Commander of the National Guard of Ukraine.

ABSTRACT The article proposes a fast algorithm for constructing the transitive closures between all pairs of nodes in the structure of a network object, which can have both directional and non-directional links. The algorithm is based on the disjunctive addition of the elements of certain rows of the adjacency matrix, which models (describe) the structure of the original network object. The article formulates and proves a theorem that using such a procedure, the matrix of transitive closures of a network object can be obtained from the adjacency matrix in two iterations (traversal) on such an array. An estimate of the asymptotic computational complexity of the proposed algorithm is substantiated. The article presents the results of an experimental study of the execution time of such an algorithm on network structures of different dimensions and with different connection densities. For this indicator, the developed algorithm is compared with the well-known approaches of Bellman, Warshall-Floyd, Shimbel, which can also be used to determine the transitive closures of binary relations of network objects. The corresponding graphs of the obtained dependences are given. The proposed algorithm (the logic embedded in it) can become the basis for solving problems of monitoring the connectivity of various subscribers in data transmission networks in real time when managing the load in such networks, where the time spent on routing information flows directly depends on the execution time of control algorithms, as well as when solving other problems on the network structures.

KEYWORDS adjacency matrix; algorithm; computational complexity of the algorithm; disjunctive addition of matrix elements; network object; transitive closure.

I. INTRODUCTION

IN his activity, a person constantly collides with objects that have a network structure. Quite often, these objects have a large number of nodes and connections between them. At the same time, individual elements of such objects can be located at a considerable distance from each other (be scattered over a large territory), which, when controlling the connectivity (reachability) of certain nodes, leads to the need to model such objects and solve some problems. One of the tasks on the network objects is the task of constructing the transitive closures (TC) of binary relations. Usually, the tasks of building the TC binary relations arise in the field of logistics, control of the topology of transport communications, development and construction of large objects based on a network structure, management of information flows in data transmission networks, building queries to distributed databases and other areas.

Today, the most well-known classical algorithms for constructing the TC binary relations include the algorithms of Bellman [1], Warshall-Floyd [2-6], Shimbel [7], Purdom [8], Depth-first search (DFS) [9, 10].

The rapid development in the second half of the 20th century of computing systems and the need to manage databases became an impetus in the development of

specialized algorithms for constructing the TC. To solve these problems, many algorithms were developed and proposed at that time, which, in addition to constructing the TC itself, also optimize structures for representing and storing data in the memory of the computing system. These algorithms include algorithms that were presented in [11-14] and a number of others. A special attention deserves the work of H. Jagadish [15], in which the author proposed an indexing scheme that allows storing the computed TC in a compressed form, which makes it possible to reduce the size of the memory used.

Recently, the notion of TC has been increasingly used in substantiating the logic of computations based on inductive reasoning [16-18], as well as in the development of software tools for searching information on the Internet [19].

A separate direction in the development of algorithms for constructing the TC has become the technology, which is based on the search in the structure of the graph of strongly connected components, followed by the search for connections between the components. This approach was applied in the algorithms of P. Purdom [8], J. Dzikiewicz and M. Sysło [20, 21], J. Eve and R. Kurki-Suonio [22], L. Schmitz [23]. A feature of such algorithms is the fact that they work on directed graphs (structures). This feature imposes a number of restrictions on the solution of some practical problems.

Some authors use the DFS-procedure as a basic algorithm to construct the TC. From this position of view, the work of Y. Ioannidis, R. Ramakrishnan and L. Winger can be noted [24]. It should be said here that the DFS-procedure is also used in the algorithms already mentioned above, described in [8] (P. Purdom) and [10] (R. Tarjan).

The algorithms mentioned in the article, due to their narrow specialization, have functional redundancy (search of the strong component, search and identifying of the shortest patch), which, when constructing the TC binary relations, leads to increased execution time. In some areas, where the time for making a control decision is commensurate with the execution time of the algorithm, this is critical and unacceptable. Therefore, the development and application of fast algorithms for calculating the TC is a topical issue in the theory of computation.

In this article, we will show that due to the use of disjunctive addition procedures (disjunctive nesting into each other) of rows of the adjacency matrix that models the original network object, it is possible to reduce the execution time of the algorithm for constructing the TC of binary relations of network objects.

The idea of disjunctive nesting of rows of an adjacency matrix for constructing the TC binary relations in the structure of an initial arbitrary graph was first expressed in the work [25].

The algorithm proposed in this work uses two adjacency matrices S^1 and S^2 dimensions $n \times n$, where n – the number of vertices in the original graph G. The algorithm performs row-by-row traversal of the matrix S^1, and if some of its elements $s_{ij}^1 \neq 0$, then the element-wise disjunctive embedding of the *j*-th row into the *i*-th row is carried out, namely, Expression 1 is used:

$$s_{ij}^2 := s_{ij}^1 \vee s_{jk}^1, \text{ where } i, j = \overline{1,n},\ k = \overline{1,n}. \qquad (1)$$

That is, when processing a certain vertex, all arcs originating from it are processed. Processing of some arc (i,j) consists in adding to the graph G arc (i,k) for each arc (j,k), outgoing from the vertex j. Thus, breadth-first search (BFS) [26] is implemented, which is obvious from the operation of disjunctive addition of the corresponding rows. However, unlike the well-known BFS, this search is local in nature every time. The place of the beginning of the next search is determined by the number of the processed vertex and its place in the structure of the graph G (see Figure 1). We call this processing a Local Breadth-First Search (LBFS) procedure.

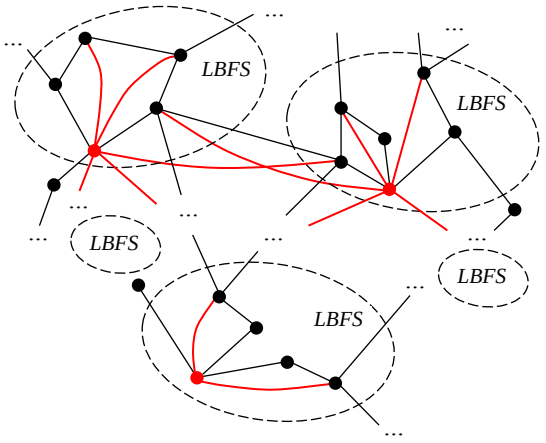

Figure 1. A fragment of the work of the *LBFS* procedure (the analyzed vertices and constructed TC are marked in red)

Results of analysis of array elements S^1 entered into S^2. After such traversal, an element-wise comparison of the arrays is performed S^1 and S^2. If they do not match, the elements of the array S^1 are rewritten into S^1 and the traversal of the array repeats. A sign of stopping the algorithm is the identity of the arrays S^1 and S^2, or doing $n-1$ iterations of the array S^1, at the same time it is asserted that, $S^1 = S^2 = R_G^+$, where R_G^+ – an array that carries information about the TC of the binary relations of the original graph G.

The computational complexity of such an algorithm is of the order of $O(n^4)$, since n^3 addition and assignment operations are performed on each traversal (see Expression

1), and such traversals in the asymptotical must be performed $n-1$ times.

However, in this article we will show that TC of binary relations of the original graph (array R_G^+) can be constructed: firstly, for a significantly smaller number of traverses of the original adjacency matrix (S^1); secondly, using only one array, which together leads to a decrease in the computational complexity of such an algorithm, as well as to a decrease in the used hardware resources – the memory of the computing system.

II. MATERIALS AND METHODS

A. THE CONSTRUCTION OF THE TRANSITIVE CLOUSURE OF A NETWORK OBJECT BY THE METHOD OF DISJUNCTIVE NESTING OF ROWS OF AN ADJACENCY ARRAY

The optimization of such an algorithm, at the stage of disjunctive nesting of rows, can be carried out if the terms are analyzed before disjunctive addition. Obviously, if there is some element $s_{ij}^1 = 1$ in the *i*-th line or some element $s_{jk}^1 = 0$ in the *j*-th line, then the operation described by Expression 1 is redundant. Considering that the comparison operation is faster than the addition and assignment operations, on large n, you can get a significant reduction in the execution time of the algorithm.

The results of disjunctive nesting of rows will not be entered into S^2, but accumulated in a single current array S_G. Taking into account the above corrections, we formulate and prove the following theorem.

Theorem 1. The transitive closure matrix R_G^+ of the original arbitrary graph G is calculated on the basis of the adjacency matrix S_G in two rounds (*m*=2) of the current array by disjunctive embedding into the analyzed row of those rows whose indices correspond to the column indices of nonzero elements in the analyzed row.

Proof: Consider separate sections of some initial graph G. Suppose that two arbitrary vertices s and t of the graph G are transitively closed. If so, then between them there are a number of paths connecting them. Then, applying the procedure L, on the first traversal of the adjacency array for any vertex i, a certain arc (i,k) can be added to the graph G. In this case, two situations are possible:

(a) $i > k$ or $k = t$;

(b) $i < k$ or $k = t$.

In the first case, a certain path $M_{st} = \{(s,i_1),(i_1,i_2),(i_2,i_3),...,(i_m,t)\}$ is formed in the graph G for which $s > i_1 > i_2 > i_3 > ... > i_m$. For example, path $M_{st} = \{(4,2),(2,1)\}$, see Figure 2.

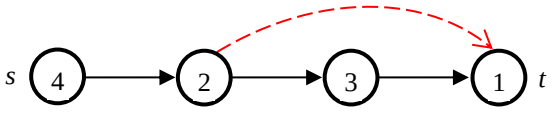

Figure 2. Formation of a section of a path with monotonically decreasing numbers of vertices

In the second case, the path will have the $s < i_1 < i_2 < i_3 < ... < i_m$ property. For example, the path $M_{st} = \{(1,3),(3,5)\}$ (see Fig. 3).

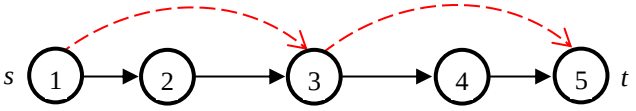

Figure 3. Formation of a track section with monotonically increasing numbers of vertices

Thus, after the completion of the first traversal of the adjacency array S_G, in the structure of the intermediate graph between two arbitrary vertices s and t, it is possible to select the path sections with monotonically decreasing and increasing numbers of vertices. Let us consider the behavior of the algorithm on such structures.

Since the processing of arcs (viewing the rows of the current array) occurs in the order of increasing numbers of the source vertices, then on the second traversal in situation (a), when processing some arc (s,i_1), all previous ones $(i_1,i_2),(i_1,i_3)...(i_1,t)$ will be taken into account, which will lead to the introduction of the desired transitive arc closure (s,t).

For example, processing an arc $(4,3)$ takes into account both the arc $(3,2)$ and the previously entered arc $(3,1)$, which adds arcs $(4,2)$ and $(4,1)$ to the desired structure, respectively (see Figure 4). The numbers indicate the order of adding arcs.

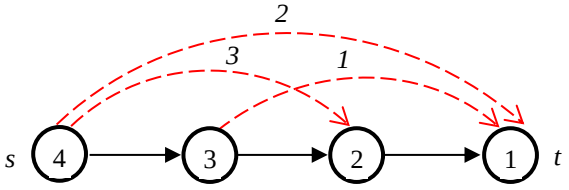

Figure 4. Construction of the TC on a section of a path with monotonically decreasing numbers of vertices

In situation (b), the arc (s,i_1) will be encountered first (see Figure 5). Its processing will cause the appearance of

the arc (s,i_2), since there is an arc (i_1,i_2) in the graph. Since $i_2 > i_1$, the arc (s,i_2) will also be processed (the corresponding unit element in the generated array R_G^+ is located to the right of the element responsible for the arc (s,i_1)). Processing it will result in the appearance of an arc (s,t) since there is an arc (i_2,t), etc.

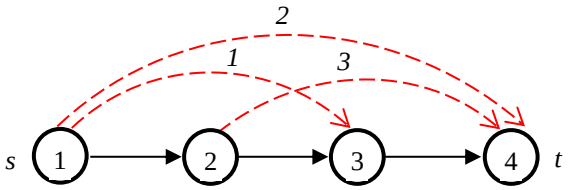

Figure 5. Construction of the TC on a section of a path with monotonically increasing numbers of vertices

Thus, such an algorithm to construct the TC of binary relations of the network object (array definition R_G^+) enough two traversals (m=2) of the stream array. The theorem is proved.

The principle of operation of the algorithm (let us call it DDAA – Double Disjunctive Addition Algorithm), based on Theorem 1, is generally described by the following Expressions 2 and 3:

$$if \quad s_{ij} \neq 0 \Rightarrow \; r_{ij}^{m=1} := s_{ij} \vee s_{jk}, \; i,j = \overline{1,n}, \; k = \overline{1,n}, \qquad (2)$$

$$r_{ij}^{m=2} := r_{ij}^{m=1} \vee r_{jk}^{m=1}, \; i,j = \overline{1,n}, \; k = \overline{1,n}, \qquad (3)$$

where m – is the number of traversal of the current matrix, n – is the dimension of the original graph G.

Obviously, the complexity of this algorithm can be estimated as ***O(n³)***.

B. AN EXAMPLE OF CONSTRUCTING THE TRANSITIVE CLOUSURE

Let the structure of an arbitrary directed graph G be given by the adjacency matrix S_G, Expression 4.

$$S_G = \begin{array}{c|ccccc} & l1 & l2 & l3 & l4 & l5 \\ \hline l1 & 0 & 0 & 0 & 0 & 1 \\ l2 & 1 & 0 & 0 & 0 & 0 \\ l3 & 0 & 1 & 0 & 0 & 0 \\ l4 & 0 & 0 & 0 & 0 & 0 \\ l5 & 0 & 0 & 1 & 0 & 0 \end{array}. \qquad (4)$$

After the completion of the first traversal (m=1) of the matrix elements by Algorithm DDAA, the intermediate array of the TC will be represented by Expression 5, and after the second traversal (m=2) by Expression 6.

$$R_G^{m=1} = \begin{array}{c|ccccc} & l1 & l2 & l3 & l4 & l5 \\ \hline l1 & 0 & 0 & 1 & 0 & 1 \\ l2 & 1 & 1 & 1 & 0 & 1 \\ l3 & 1 & 1 & 1 & 0 & 1 \\ l4 & 0 & 0 & 0 & 0 & 0 \\ l5 & 1 & 1 & 1 & 0 & 1 \end{array}. \qquad (5)$$

$$R_G^+ = R_G^{m=2} = \begin{array}{c|ccccc} & l1 & l2 & l3 & l4 & l5 \\ \hline l1 & 1 & 1 & 1 & 0 & 1 \\ l2 & 1 & 1 & 1 & 0 & 1 \\ l3 & 1 & 1 & 1 & 0 & 1 \\ l4 & 0 & 0 & 0 & 0 & 0 \\ l5 & 1 & 1 & 1 & 0 & 1 \end{array}. \qquad (6)$$

The algorithm will stop after the second traversal, and the array $R_G^{m=2}$ will correspond to the desired array R_G^+, which describes the TC binary relations in the original graph G. A further increase in the number of traversals (m=3) will not lead to a change in the array $R_G^{m=2}$, which confirms the theorem proved above and the correctness of the developed algorithm.

C. SOFTWARE IMPLEMENTATION

Considering the above, the DDAA algorithm proposed in this article can be implemented as the following procedure:

```
procedure TForm1.DDAA;
var i, j, jj, n: Integer;
              m: Short;
begin
          for m:=1 to 2 do
          for i:=0 to n-1 do
          for j:=0 to n-1 do
              If (S[i,j]=1) and (i<>j) then
              begin
              for   jj:=0   to   n-1   do   If
(S[i,jj]=0) and (S[j,jj]=1) then S[i,jj]:=1;
              end;
end.
```

III. RESULTS AND DISCUSSION

A. THE EXPERIMENTAL ESTIMATION OF THE COMPUTATIONAL COMPLEXITY OF THE ALGORITHM

In the course of the experiment, the dependence of the execution time of various algorithms on the dimension of the network object (test task) is established as well as the density of the edges in it. In particular, this approach was applied and described in [27, 28]. According to this indicator, the developed DDAA algorithm was compared with the well-known classical algorithms of Bellman (computational complexity $O(n^3)$ [29]), Warshall-Floyd (computational complexity $O(n^3)$ [29]) and Shimbell (computational complexity $O(n^4)$ [7]). The choice of these algorithms is not accidental, since they are used in practice, due to its low computational complexity and ease of implementation. At the same time, it was noted in [7] that the most efficient is the Warshall-Floyd algorithm, appropriately programmed.

For the correctness of the experiment, all four algorithms were programmed by one programmer. Programming environment was Delphi 7.0, Object Pascal language. The experiment was carried out on the same computer.

B. THE EXPERIMENT LAYOUT AND RESULTS

To plot the dependency graphs, the sizes of test tasks were changed in the range $n=\overline{10,100}$ with a step of 10, where n – the number of vertices of the graph that models the network object. At each step, the time spent by each investigated algorithm on solving 1000 test tasks was recorded. Next, the average time spent on solving the one task by each algorithm was calculated.

The input data was formed in two variants:

(a) at each step, the same test task of the corresponding dimension n (stationary task) was used;

(b) at each step, a set of the different test tasks of the corresponding dimension n was used. The density (number) of nonzero elements in the original matrix S_G was set by a generator of random integers from the set $\{0,...,n^2\}$ – (dynamic tasks).

The curves were built in the mathematical package MathCAD from the points obtained during the experiment.

The advantages of the proposed DDAA algorithm over the Warshall's algorithm when forming the input data according to variant (a) for dimensions n=30 reach about 8%, and for n=100 already 14% (see Figure 6). The advantage over the Bellman's algorithm is 35...50% for all dimensions that were considered during the experiment. With the input data generated according to variant (b), the execution time of the DDAA algorithm is less than that of the Warshall algorithm by an average of 45...60%, depending on the dimension of the test task. Approximately the same indicators are retained in comparison with the Bellman's algorithm.

Figure 6 presents the dependences obtained for discharged, but coupled structures. That is, the corresponding data arrays were characterized by 6...10% content of nonzero elements of the total number of elements in the array.

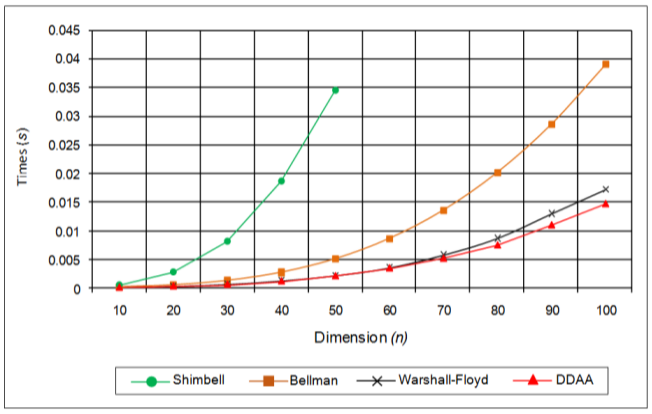

Figure 6. The graph of the dependence of the execution time on the dimension of stationary tasks, (variant (a))

When solving dynamic tasks (variant (b)), the execution time of all algorithms increased (see Figure 7). A sharp increase is observed in the Warshall algorithm. This is explained by the fact that in case (b) the input data sets were denser and were characterized on average by 45...55% content of nonzero elements, the analysis of which the Warshall algorithm spent more time on.

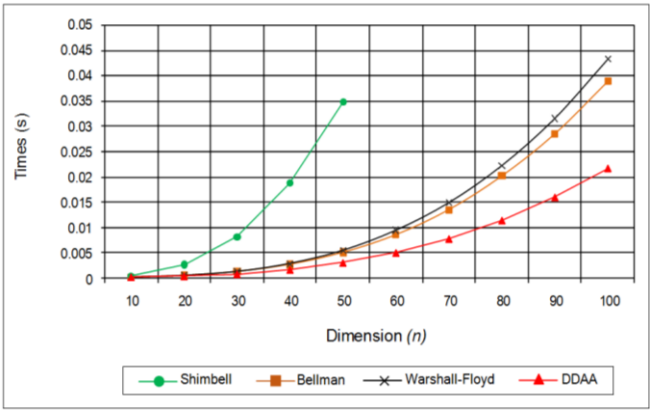

Figure 7. The graph of the dependence of the execution time on the dimension of dynamic tasks, (variant (b))

Bellman's and Shimbel's algorithms differ in terms of execution time both under the conditions of variant (a) and variant (b), since they do not contain conditional operators, and therefore, without analyzing the values of array elements, they execute three nested loops. However, the execution time of these algorithms is longer than that of other algorithms.

The logic that is implemented in DDAA algorithm assumes the execution of conditional operators, however, due to this, the algorithm does not perform redundant operations of disjunctive nesting of rows of the adjacency array, and does not perform unnecessary assignment operations, which makes it more efficient. The increase in

the execution time for DDAA algorithm on different dimensions of test tasks during the formation of input data arrays for variant (b) in comparison with variant (a) was 14...30%. For the Warshall algorithm, this indicator was 31...58%.

IV. CONCLUSIONS

Theoretical research in the field of connectivity of network objects made it possible to formulate and prove the theorem on the possibility of constructing the TC of binary relations of network objects by disjunctive nesting of the corresponding rows of an adjacency array in two rounds of such an array.

The DDAA algorithm developed on the basis of this theorem is accurate, and its computational complexity is less than that of the known analogs and in asymptotics is of the order of $O(n^3)$.

The advantages of DDAA algorithm over its closest competitors (Bellman's and Warshall's algorithms) on all considered dimensions of test tasks generated by variant (b), on average, are 1,8...2 times. The nature of the trends presented in Figure 7 allows us to assert that this pattern will also persist on network objects with a higher dimension than the one that was selected to be the maximum (n=100) during the experiment.

The algorithm works with both undirected and oriented structures which makes it versatile.

The proposed algorithm can find its application in solving adaptive routing problems in data transmission networks when managing incoming load to eliminate blockages in such networks [30], where the time spent on routing information flows (individual packets) directly depends on the execution time of link state algorithms (*Link State Algorithms*, LSA). In addition, LSAs must be fairly simple, not requiring a lot of computational resources.

We consider that the DDAA algorithm proposed in the article fully meets all these requirements.

References

[1] R. Bellman, "On a Routing Problem," *Quarterly of Applied Mathematics*, vol. 16, issue 1, pp. 87-90, 1958. https://doi.org/10.1090/qam/102435.

[2] R. Floyd, "Algorithm 97: Shortest path," *Communications of the ACM*, vol. 5, issue 6, 345 p., 1961, https://doi.org/10.1145/367766.368168.

[3] S. Hougardy, "The Floyd-Warshall algorithm on graphs with negative cycles," *Information Processing Letters*, vol. 110, no. 8-9, pp. 279-281, 2010. https://doi.org/10.1016/j.ipe.2010.02.001.

[4] S. Warshall, "Algorithm on Boolean matrices," *Journal of the ACM*, vol. 9, issue 1, pp. 11-12, 1962. https://doi.org/10.1145/321105.321107.

[5] H. Warren, "A modification of Warshall's algorithm for the transitive closure of binary relations," *Commun. ACM*, vol. 18, issue 4, pp. 218-220, 1975. https://doi.org/10.1145/360715.360746.

[6] Z. Ding, W. Shu and M. Wu, "FPGA based parallel transitive closure algorithm," *Proceedings of the 2011 ACM Symposium on Applied Computing SAC'11*, March 2011, pp. 393-394. https://doi.org/10.1145/1982185.1982270.

[7] A. Shimbel, "Structural parameters of communication networks," *Bulletin of Mathematical Biophysics*, vol. 15, issue 4, pp. 501-507, 1953. https://doi.org/10.1007/BF02476438.

[8] P. Purdom, "A transitive closure algorithm," *Bit 10*, vol. 1, 1970, pp. 76-94. https://doi.org/10.1007/BF01940892.

[9] T. Cormen, C. Leiserson, R. Rivest and C. Stein, *Introduction to Algorithms*, Second Edition. Section 22.3: Depth-first search, MIT Press and McGraw-Hill, 2001, pp. 540-549.

[10] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J. Computing*, vol. 1, issue 2, pp. 146-160, 1972. https://doi.org/10.1137/0201010.

[11] R. Agrawal, S. Dar and H. Jagadish, "Direct transitive closure algorithms: Design and performance evaluation," *ACM Trans. Database Syst.*, vol. 15, issue 3, pp. 427-458, 1990. https://doi.org/10.1145/88636.88888.

[12] Y. Chen, "On the graph traversal and linear binary-chain programs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, pp. 573-596, 2003. https://doi.org/10.1109/TKDE.2003.1198392.

[13] S. Dar, R. Ramarkrishnan, "A performance study of transitive closure algorithm," *Proceedings of the SIGMOD International Conference*, Minneapolis, Minnesota, USA, 1994, pp. 454-465. https://doi.org/10.1145/191843.191928.

[14] A. Velasquez, S. Jha, "Brief announcement: Parallel transitive closure within 3D crosspoint memory," *Proceedings of the 30th Symposium on Parallelism in Algorithms and Architectures SPAA '18*, July 2018, pp. 95-98. https://doi.org/10.1145/3210377.3210657.

[15] H. Jagadish, "A compression technique to materialize transitive closure," *ACM Trans. Database Systems*, vol. 15, issue 4, pp. 558-598, 1990. https://doi.org/10.1145/99935.99944.

[16] L. Cohen, R. N. S. Rowe, "non-well-founded proof theory of transitive closure logic," *ACM Transactions on Computational Logic*, vol. 21, issue 4, Article no. 31, pp. 1-31, 2020. https://doi.org/10.1145/3404889.

[17] W. Charatonik, E. Kieroński and F. Mazowiecki, "Decidability of weak logics with deterministic transitive closure," *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL'14) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'14)*, July 2014, Article no. 29, pp. 1-10. https://doi.org/10.1145/2603088.2603134.

[18] A. Eriksson, P. Jansson, "An agda formalisation of the transitive closure of block matrices (extended abstract)," *Proceedings of the 1st International Workshop on Type-Driven Development TyDe'2016*, September 2016, pp. 60-61. https://doi.org/10.1145/2976022.2976025.

[19] H. Yin, A. Benson and J. Leskovec, "the local closure coefficient: a new perspective on network clustering," *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining WSDM'19*, January 2019, pp. 303–311. https://doi.org/10.1145/3289600.3290991.

[20] J. Dzikiewicz, "An algorithm for finding the transitive closure of a digraph," *Computing*, vol. 15, pp. 75-79, 1975. https://doi.org/10.1007/BF02252839.

[21] M. Sysło, J. Dzikiewicz, "Computational experiences with some transitive closure algorithms," *Computing*, vol. 15, pp. 33-39, 1975. https://doi.org/10.1007/BF02252834.

[22] J. Eve, R. Kurki-Suonio, "On computing the transitive closure of a relation," *Acta Informatica*, vol. 8, pp. 303-314, 1977. https://doi.org/10.1007/BF00271339.

[23] L. Schmitz, "An improved transitive closure algorithm," *Computing*, vol. 30, pp. 359-371, 1983. https://doi.org/10.1007/BF02242140.

[24] Y. Ioannidis, R. Ramakrishnan and L. Winger, "Transitive closure algorithms based on graph traversal," *ACM Trans. Database Syst.*, vol. 18, vol. 3, pp. 512-576, 1993. https://doi.org/10.1145/155271.155273.

[25] A. Kuznetsov, *Mathematical Modeling of Design Objects in ADS*, Kharkov Military University Press, Kharkov, Ukraine, 1994, 92 p.

[26] C. Lee, "An algorithm for path connections and its applications," *IEEE Transactions on Electronic Computers*, vol. 10, no. 3, pp. 346-365, 1961. https://doi.org/10.1109/TEC.1961.5219222.

[27] K. Hanauer, M. Henzinger and C. Schulz, "Faster fully dynamic transitive closure in practice", Proceedings of the 18th International Symposium on Experimental Algorithms (SEA'2020), 2020, Article no. 14; pp. 1-14. https://doi.org/10.4230/LIPIcs.SEA.2020.14.

[28] V. Pieterse, L. Cleophas, "Benchmarking optimized algorithms for transitive closure," *Proceedings of the South African Institute of Computer Scientists and Information Technologists SAICSIT'17*,

September 2017, vol. 27, pp. 1-10. https://doi.org/10.1145/3129416.3129425.
[29] V. Lipskyy, *Combinatory for Programmers*, Moscow, Mir Press., 1988, 213 p. (in Russian)
[30] R. Al-Dujaily, T. Mak, F. Xia, A. Yakovlev and M. Palesi, "Embedded Transitive Closure Network for Runtime Deadlock Detection in Networks-on-Chip", *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, issue 7, pp. 1205-1215, 2012. https://doi.org/10.1109/TPDS.2011.275.

VLADIMIR BATSAMUT, Doctor of Military Science, Professor. Current position a Deputy Head of the Research Center of the National Academy of the National Guard of Ukraine, Kharkiv. His research interests include justification of decisions in the military sphere, optimization of network objects and control their condition, algorithmization of processes and software development, analysis of data and forecasting the development of processes.

SVIATOSLAV MANZURA has PhD in Weapons and Military Technology. Current position a Head of the Research Laboratory, Research Center of the National Academy of the National Guard of Ukraine, Kharkiv. His research activities are focused on research qualitative and quantitative characteristics of technical systems, information support and analysis of data and processes.

OLEKSANDR KOSIAK is Master of Military Administration. Current position a graduate student of the National Academy of the National Guard of Ukraine, Kharkiv. His research activities are focused on data processing, justification of decisions in the military sphere.

VIACHESLAV GARMASH. Current position a Senior Researcher, Research Laboratory, Research Center of the National Academy of the National Guard of Ukraine, Kharkiv. His research interests are in the field of security of computer systems and data transmission networks.

DMYTRO KUKHARETS is Master of Military Administration. Current position a Senior Researcher, Research Laboratory, Research Center of the National Academy of the National Guard of Ukraine, Kharkiv. His research interests are in the field of security of computer systems and optimization of processing of big data.