UDC 004.82+004.94(045)
DOI:10.18372/1990-5548.85.20427

[1]M. Z. Zgurovsky,
[2]A. O. Boldak,
[3]K. V. Yefremov,
[4]V. M. Statkevych,
[5]O. A. Pokhylenko

A FORMAL LANGUAGE FOR THE ANALYSIS OF GRAPH MODELS AND ITS SOFTWARE IMPLEMENTATION

National Technical University of Ukraine "Ihor Sikorsky Kyiv Polytechnic Institute," Kyiv, Ukraine
E-mails: [1]mzz@kpi.ua ORCID 0000-0001-5896-7466,
[2]boldak@wdc.org.ua ORCID 0000-0001-7151-0743,
[3]k.yefremov@wdc.org.ua ORCID 0000-0003-3495-6417,
[4]mstatkevich@yahoo.com,
[5]o.pokhylenko@kpi.ua ORCID 0000-0002-1562-2051

***Abstract*—*The purpose of this paper is to develop a specialized language for processing graph data and its software implementation. The proposed solution ensures versatility, usability, and efficiency, enabling the execution of both basic graph operations and more complex procedures. The tool supports classical graph algorithms, including shortest path search, graph traversal, and minimum spanning tree construction, as well as applications in modeling and analyzing message transition processes and processing graph-based representations of textual data. A common drawback of many existing graph analysis tools—often implemented as libraries of general-purpose programming languages—is their limited usability. This limitation arises from the fact that the description of graph analysis procedures relies on data structure operations defined in terms of these general-purpose languages, which complicates perception and reduces the clarity of the mathematical methods being implemented. Developing a specialized domain-specific language based on high-level abstractions can address these shortcomings. Such a language will provide a formalized description of methods for analyzing and processing graph models, improving their comprehensibility and accessibility to users. Its software implementation will deliver ready-to-use solutions for executing graph analysis methods. Composing such methods will facilitate solving a wide range of tasks, including the analysis of natural language messages and the study of information publication and dissemination processes in online environments.*

Keywords—Graph models; language for analysis of graph models; semantic networks; syntactical analysis; message transition; rumor spreading process.

I. INTRODUCTION

Graph models are an essential tool in many areas of modern science where it is necessary to model and analyze relationships between objects. Representing complex systems in the form of nodes and edges makes it possible to effectively study both individual objects (system components) and their interconnections, which makes graph models indispensable for analyzing a wide range of systems.

Graph models are widely used for routing and flow optimization in computer networks and transportation systems. Tasks in these domains employ algorithms such as shortest path search, maximum flow computation, and minimum spanning tree construction.

In the field of intelligent text data analysis, graph models are applied to build semantic networks derived from unstructured texts. This area covers various types of graph models used to address tasks such as content comparison, detection of inconsistencies and contradictions, news aggregation, digest creation, and text summarization.

Graph models have also been employed in the analysis of message transition processes across various networks. They are used to model the spread of information, news, viral content, and even epidemics. Such models make it possible to assess the speed, scale, and patterns of propagation.

Over the past two decades, numerous tools have been developed to support working with graph models and performing their analysis. Examples include Neo4j, which uses the Cypher Query Language (Cypher QL), SPARQL, and others. These software solutions provide convenient means for the graphical representation of data and powerful query languages that share certain similarities with SQL while differing from it due to their specialization in graph structures.

II. Rule-Based Approaches to Building Semantic Networks

Semantic networks can be constructed through rule-based extraction of semantic relations. In article [1], the following rule-based approaches are considered:

- lexical analysis using phrase patterns;
- text analysis using document structure;
- syntactical analysis using dependency structure.

Each of these approaches is briefly described below.

Lexical analysis applies predefined patterns to text fragments containing target entities in order to extract relations between these entities.

Text analysis using document structure takes into account elements such as the document title, headlines, and so forth when extracting relations. In article [1], this approach is discussed in the context of regulatory documents, where it proved to be critical due to the presence of references to other documents and sections within the same document.

Syntactical analysis employs additional information about dependencies between words, such as Universal Dependencies [2] and coreference [3]. In this case, rules are formulated for extracting semantic relations from word dependency graphs in sentences. Syntactical analysis enables the detection of negations and allows relation extraction to be extended to longer text spans involving coreference.

Since some types of texts – such as news articles and social media posts – lack complex structure, applying document-structure-based text analysis may be impractical. Therefore, this paper focuses on syntactical analysis with the use of dependency structures as the most promising approach.

Two aspects of syntactical analysis are considered in more detail.

First, syntactical analysis using word dependency structure (dependency parsing) requires prior morphosyntactic annotation of the text. As a result of this preprocessing, each sentence can be represented either as a directed Universal Dependencies tree [2] or as a directed graph of Enhanced Universal Dependencies [4]. A dependency is treated as a binary asymmetric relation, where each dependency has a grammatical relation label; for basic Universal Dependencies, there are 37 such labels. Universal Dependencies organize the words of a sentence into a tree structure with the main predicate as the root. However, in the case of Enhanced Universal Dependencies, additional links mean that the resulting graphs are not necessarily trees. A distinctive feature of Universal Dependencies is their applicability to cross-linguistically consistent morphosyntactic annotation – [2] reports that they have been used for more than 100 languages. It is also worth noting that certain text analytics tools, such as Stanford CoreNLP (Java) [5] and Stanza (Python) [6], include modules for automated text processing that can produce Universal Dependencies for multiple languages.

Second, an important task in syntactical analysis is coreference resolution – the task of identifying all expressions in a text that refer to the same entity. According to [3], some modern coreference resolution systems are based on multiple LLM calls; however, such systems can be prohibitively expensive for many use cases, for example, when processing large text collections. Word-level coreference resolution systems are identified separately; these are more computationally efficient but slightly less accurate. The conjunction-aware word-level coreference resolution system (CAW-coref) [3] is used in Stanza.

In the direct construction of a semantic network, relation extraction can be performed in the form of triples (subject, relation, object) for binary relations, or tuples (argument_0, relation, argument_1, …, argument_n) for n-ary relations [7]. In the Open Information Extraction (OpenIE) task, the use of triples is most common – here, the subject and object correspond to nodes of the semantic network, while the relation corresponds to an edge between the respective nodes. In article [8], extraction of triples is performed by splitting each long sentence in the text into short, coherent clauses and then identifying the simplest justified relation triples for each of them. In this approach, dependency-based rules (dependency patterns) are used for relation extraction. It is noted that traditionally, relation triples have been extracted using large rule sets; however, this method is fragile with respect to out-of-domain text and long-distance dependencies and does not capture the substructure of arguments. Instead of a large rule set for extracting relations from complex sentences, [8] employs a small set for canonically structured sentences along with a classifier trained to extract self-contained clauses from longer sentences.

III. Approach to Developing a Domain-Specific Language for Processing and Analyzing Graph Models

To ensure that the developed domain-specific language can be applied to a wide range of tasks related to graph model analysis, it must provide at least a basic set of commands for manipulating

graphs. This core set should include commands for performing elementary operations on graphs, such as adding, modifying, and deleting nodes and edges.

The functions enabling graph manipulation correspond to the basic set of commands described in the following section.

Let us consider an arbitrary graph $G=(V,E)$, where V is the set of nodes (vertices) and E is the set of edges. According to the commands of the developed language, several types of functions are defined; we will examine three of them in detail.

A projector function $Pr_V : G \to V$, extracts the set of nodes V from a graph G. The domain of the projector function $D(\mathrm{Pr}_V) = G$ is the set of graphs of the given form, and the range $E(\mathrm{Pr}_V) = V$ is the set of nodes. Similarly, the projector function $\mathrm{Pr}_E : G \to E$ extracts the set of edges E from a graph G, with the domain $D(\mathrm{Pr}_E) = G$ being the set of graphs, and the range $E(\mathrm{Pr}_E) = E$ being the set of edges.

An aggregator function $Aggr : V \times E \to G$ constructs a graph from a given set of nodes V and a set of edges E. The domain of the aggregator $D(Aggr) = V \times E$ is the pair of sets V and E, and the range $E(Aggr) = G$ is the set of graphs.

A set of transformer functions $Tr_i : S_i \times P \to S_j$ allow modification of a set S_i to obtain a set S_j according to a given set of parameters P. The parameters P specify how the set S_i should be transformed. For example, to transform a set of nodes V by adding, modifying, or deleting a specified node, we can define the transformer $Tr_V : V \times P \to V$, where the domain $D(Tr_V) = V \times P$, consists of the set of nodes V and the set of parameters $P = \{\text{add, modify, delete}\}$, while the range $E(Tr_V) = V$, is the updated set of nodes.

These commands enable the addition and deletion of arbitrary nodes, as well as the addition and deletion of edges between any two nodes, thereby supporting the creation and modification of any graph of the specified form. For example, to add a new node v and a new edge e to a given graph G:

$$G + v + e = Aggr\left(Tr_V\left(\mathrm{Pr}_V(G), addv\right), Tr_e\left(\mathrm{Pr}_E(G), adde\right)\right),$$

An examination of the commands in the developed language through the lens of the proposed set of functions confirms that the language offers extensive capabilities.

IV. CORE COMMAND SET

The developed language for the analysis of graph models currently comprises three command groups: core commands, graph import / export commands, and node / edge processing commands.

The core command set includes the following:

– transform – enables the definition of an arbitrary JavaScript function and its application to a specified variable or JSON file field (corresponds to a transformer function of the form $Tr_i : S_i \times P \to S_j$);

– value – an assignment command that allows the creation of new variables or the updating of existing ones (a transformer function with a restricted parameter set $P_{\text{value}} = \{add\}$);

– delete – removes variables or object fields (a transformer function with $P_{\text{delete}} = \{\text{delete}\}$);

– forEach and while – loop constructs ($\text{forEach}(F,s) = \{F(s) \mid s \in S\}$, where F is an arbitrary function executed for each element s of set S);

– if – a conditional execution construct ($\text{if}(S,P,F_{then},F_{else}) = \begin{cases} F_{then}(S), P(S) = \text{true}, \\ F_{else}(S), P(S) = \text{false}, \end{cases}$ where P is a predicate, and F_{then} and F_{else} are functions executed depending on the evaluation of the condition).

The graph import / export commands support importing and exporting graphs in various formats (e.g., ECharts or Graphology). They also allow specifying custom parsing or formatting functions. Graph import can be formalized as an aggregator function $Aggr : V_{imp} \times E_{imp} \to G_{imp}$, while export can be described as applying two projector functions $Pr_V : G_{\text{exp}} \to V_{\text{exp}}$ and $\mathrm{Pr}_E : G_{\text{exp}} \to E_{\text{exp}}$ to a graph G_{exp}. Format conversion can be viewed as applying transformer functions to the sets of nodes and edges.

The node and edge processing command set includes forEach, map, filter, reduce, find, some, and every. These commands may include additional constraints, such as restricting execution to the neighbors of a specific node or to the incoming / outgoing edges of a given node in a directed graph. The associated functions are as follows:

$$\text{map}(F,S) = \{F(S) \mid s \in S\},$$

$$\text{filter}(P,S) = \{s \mid s \in S, P(s) = \text{true}\},$$

$$\text{reduce}(F, S, \text{acc}) = \begin{cases} \text{acc}, S = \varnothing, \\ F\left(\text{reduce}\left(F, S', \text{acc}\right), s_n\right), \end{cases}$$

$$\text{find}(P, S) = \begin{cases} s, \exists s \in S : P(s) = \text{true}, \\ \text{undefined}, \end{cases}$$

$$\text{some}(P, S) = \begin{cases} \text{true}, \exists s \in S : P(s) = \text{true}, \\ \text{false}, \end{cases}$$

$$\text{every}(P, S) = \begin{cases} \text{true}, \forall s \in S : P(s) = \text{true}, \\ \text{false}, \end{cases}$$

where S denotes a set of nodes or edges, F is a transformer function P is a predicate (a projector function with a codomain $E(P) = \{\text{true}, \text{false}\}$), and $S' = S \setminus \{s_n\}$ with s_n being the last element of S. Additional constraints can be formalized as projector functions applied to node and edge sets prior to executing the selected operation.

It is worth noting that, in the developed software implementation, the command sets are provided as plugins. This design allows for the addition of more sophisticated and useful commands to the interpreter when necessary. Furthermore, flexibility is enhanced by the ability to define arbitrary transformations and store functions in variables. This feature enables the creation of reusable program modules that can be incorporated into scripts as needed.

A. Software Implementation

Usability is a crucial aspect of the language for analyzing graph models. A simple syntax enables users without deep technical expertise to effectively utilize the language and facilitates their learning process. The YAML syntax is minimalist and highly readable, making it well-suited for use in configuration files and rule specification. An example of a language that is a subset of YAML and is employed for constructing knowledge graphs or semantic networks based on rules is YARRRML [9], [10].

Since YAML is designed to support single-pass processing, scripts written in YAML can be interpreted without prior compilation. Consequently, the decision was made to develop an interpreter for the graph model analysis language. Considering the importance of cross-platform compatibility, the interpreter was implemented in TypeScript. Programs written in TypeScript can be compiled into JavaScript, enabling the developed interpreter to be executed both on servers using the Node.js platform and in virtually any modern web browser.

The implemented software processes two types of input files: JSON files containing initial graph data, and YAML scripts specifying transformation rules for these graphs. When YAML scripts are interpreted by the developed program, corresponding function calls to the Graphology library [11] for graph manipulation are executed. Upon completion of script execution, the output is a new JSON file reflecting the transformed graph. The resulting graph can then be visualized using the Apache ECharts library [12].

B. Use of the MOLFAR GAL Language for Solving Tasks of Graph-Based Text Data Representation

To demonstrate the capabilities of the developed language, we consider several tasks. Let us take a Universal Dependencies (UD) tree obtained from Stanza for the following simple sentence: “The President signed the new agreement, while the secretary organized meetings.” Using the developed language, we implement rules for extracting semantic relations in the form of triplets (subject, relation, object). Consider the simplest case: identifying subtrees with a verb node (VERB) as the root and two child nodes (subject and object) connected to the root by dependency edges labeled *nsubj* and *obj*, respectively.

Below is a script written in the developed language that implements the corresponding rule for semantic relation extraction:

```
- import_graph: # import directed graph
    from:      # from the data field
      $: data   # into variable graph
    type: directed
    into: graph

- filterNodes: # find verb nodes
    as: $node  # and save them into
    from:     # the verbs array
      $: graph
    where: $node.category == 'VERB'
    into: verbs
    resolve_nodes: true

- forEach: # main loop for each verb node
    item: $verb
    from:
      $: verbs
    do:
      - value:      # initialize variable $out
          set: () => ({}) # with an empty object
          into: $out
      - reduceOutEdges: # save outgoing edges
          of:          # and connected nodes
            $: $verb    # for the node
          as:          # considered in the
            edge: $edge # current iteration
            target: $target
          from:
            $: graph
          initialValue:
            $: $out
          reduce: $out[$edge.value] = $target.key, $out
          into: $out
      - if: # if $out contains dependencies
          condition: > # nsubj and obj
            'nsubj' in $out && 'obj' in $out
```

```
      then:
        - transform: # create a new edge
            from:    # labeled with the verb
              $: $out # under consideration
            action: >
              ($out, {graph, $verb}) =>
                graph.addEdge($out.nsubj, $out.obj, { value:
$verb.name })

- filterNodes: # find all nodes that are not
    as: $node  # nouns (NOUN), including
    from:      # proper nouns (PROPN)
      $: graph
    where: $node.category != 'NOUN' && $node.category !=
'PROPN'
    into: not_nouns

- forEach: # for each node in not_nouns:
    item: $word
    from:
      $: not_nouns
    do:
      - transform: # delete the node
          from:
            $: $word
          action: >
            ($word, { graph }) => graph.dropNode($word)

- export_graph: # export graph in ECharts format
    from:     # for visualization
      $: graph
    format: ECharts
    into: chart
```

The initial dependency tree, where dependencies are represented as edges and words as nodes (node labels are lemmas of words, and colors indicate parts of speech), as well as the result of executing the script, where semantic relations are represented as edges and words as nodes, are shown in Figs 1 and 2.

C. Use of the MOLFAR GAL Language for Shortest Path Search

The developed language can also be applied to classical graph theory problems, such as finding the shortest path. Below is a script implementing a simple version of the Bellman-Ford algorithm:

```
- import_graph: # import graph
    from:
      $: data
    type: directed
    into: graph

- forEachNode: # initialization: set the
    as: $node  # initial path length value to
    from:      # infinity for all nodes except
      $: graph # node A (the node from which
    do: >      # shortest paths are to be found)
      $node.value = $node.name == 'A' ? 0 : Infinity # for node A,
the value is set to 0

- value:   # define the maximum number of
    set: > # iterations (no more than number
      ({ graph }) => Array.from({length: graph.order - 1}, (_, i) => i) #
of nodes -1)
    into: iterations

- forEach: # main loop (up to the maximum
    from:  # number of iterations)
      $: iterations
    do:
      - forEachEdge:       # for each edge,
          as:              # perform relaxation
            target: $target # along the edge –
            edge: $edge # attempt to improve
            source: $source # the path length
          from:     # value for the target node
            $: graph
          do: >
            $target.value = Math.min($target.value, $source.value +
$edge.value)

- export_graph: # export graph
    from:
      $: graph
    format: ECharts
    into: chart
```

The result of the script execution is shown in Fig. 3, where the start node from which shortest paths were calculated is highlighted in blue, and all other nodes are marked in green. Each node is labeled with a letter (node identifier) and a number – the length of the shortest path from node A to the corresponding node. The graph edges are annotated with their weight values.

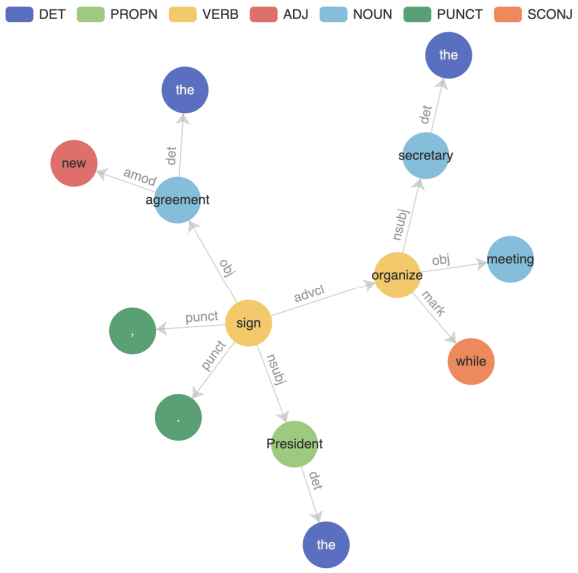

Fig. 1. Dependency tree between words in the sentence

Fig. 2. Extracted semantic relations

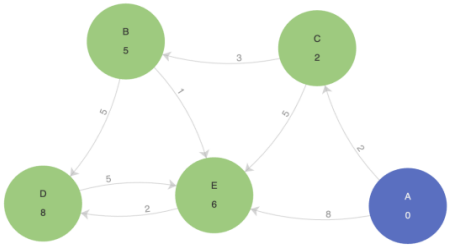

Fig. 3. Shortest path lengths from node A

D. Use of the MOLFAR GAL Language for Modeling the Message Spreading Process

We also consider the application of the developed language for modeling the process of message dissemination. A simple synchronous push model of rumor spreading is implemented on a random connected graph with $V = 25$ nodes and $E = 50$ edges. Initially, one informed node v_0 is selected, which will spread the rumors. At each iteration, all informed nodes I ($v_0 \in I$) attempt to spread the rumor (message) to their neighbors with a probability $p = 0{,}75$. If a node receives the message for the first time, it changes its state to informed (is added to the set of informed nodes I) and will start spreading the rumor from the next iteration. If a node has already received the message (is already informed), its state remains unchanged. The process continues until all nodes become informed. The script for modeling this process is given below:

```
- import_graph: # import graph
    from:
      $: data
    type: undirected
    into: graph

- value: # add node with index 0 to the set of
    set: ({ graph }) => new Set([graph.nodes()[0]]) # informed
nodes
    into: informedNodes

- value: # initialize iteration counter
    set: () => 1 # (starting value: 1)
    into: iteration

- while: # loop while the number of informed
    condition: informedNodes.size < graph.order # nodes is less
than the total number
    do: # of nodes: create a temporary set for
      - value:              # nodes informed
          set: () => new Set() # in the current
          into: newlyInformedNodes # iteration
      - forEach: # loop through each
          as: $informedNode # informed node
          from:
            $: informedNodes
          do:
            - reduceNeighbors: # for all
                of:      # neighbors of the
                  $: $informedNode # current
                as: $neighbor    # node, with
                from: # probability 0.75, the
                  $: graph # rumor is passed
                initialValue:
                  $: newlyInformedNodes
                reduce: >
                  Math.random() < 0.75 &&
newlyInformedNodes.add($neighbor.key),
                  newlyInformedNodes
                into: newlyInformedNodes
      - transform: # update the set of informed
          from:    # nodes by adding nodes
            $: informedNodes # informed this
          action: >         # iteration
            (informedNodes, { newlyInformedNodes }) =>
              new Set([...informedNodes, ...newlyInformedNodes])
          into: informedNodes
      - forEach: # loop through nodes informed
          as: $newlyInformedNode # in the
          from: # current iteration
            $: newlyInformedNodes
          do:
            - transform: # assign the
                from:    # iteration when
                  $: graph # the node first
                action: > # received a message
                  (graph, { $newlyInformedNode, iteration }) =>
                    graph.updateNodeAttribute($newlyInformedNode,
'value',
                      val => val ?? iteration)
      - transform: # increment iteration
          from:    # counter
            $: iteration
          action: (iteration) => iteration + 1
          into: iteration

- export_graph: # export graph
    from:
      $: graph
    format: ECharts
    into: chart
```

The result of the script execution is shown in Fig. 4. Each node is labeled with a letter – its unique identifier – and a numerical value indicating the iteration at which the node received the message (i.e., first heard the rumor). Nodes are colored according to the iteration number, with each numerical value corresponding to a distinct color. Node A is the initial node where rumor spreading began, hence its iteration number is 0.

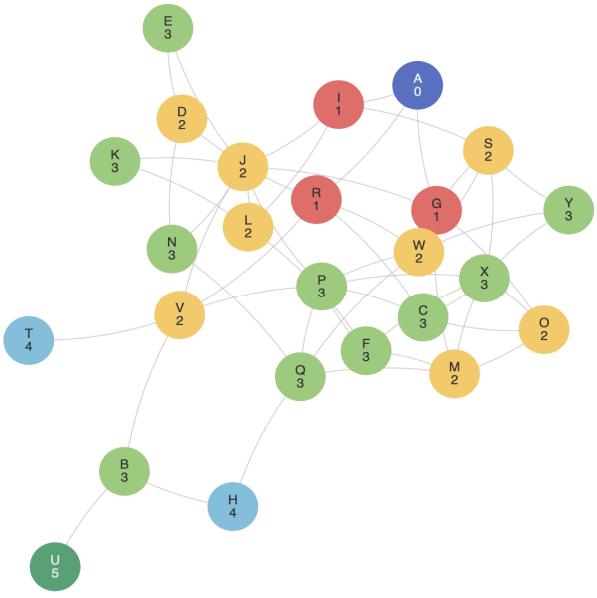

Fig. 4. Example of modeling the rumor spreading process

V. CONCLUSION

The developed language is aimed at solving a wide range of applied tasks related to the analysis of graph models. This article demonstrated its application for the following problems: extraction of semantic relations based on rules using dependency structures between words in sentences, shortest path search in directed graphs, and modelling the process of message dissemination.

In the software implementation, command sets are presented as plugins, enabling easy extensibility of the interpreter's functionality with new commands. In the future, it is planned to add commands for working with multiple graphs, including operations such as intersection, union, and other graph manipulations. Special attention will be given to expanding the functionality for processing graph representations of textual data.

The following new commands are planned for implementation:

– a command for merging multiple textual nodes into one, allowing formation of phrase-nodes based on word-nodes;

– a command for creating relation-edges from nodes, enabling identification of action nodes and their transformation into edges;

– a command for converting a semantic network into a textual format.

Besides adding new commands, development of an editor with syntax autocomplete and other usability-enhancing tools is anticipated. These improvements will not only expand the capabilities of the software solution but also make it more accessible to a wider range of users.

VI. ACKNOWLEDGMENTS

The research was supported by a grant from the National Research Foundation of Ukraine (project registration number 2023.04/0087).

REFERENCES

[1] A. Korger, & J. Baumeister, (2021, September). Rule-based Semantic Relation Extraction in Regulatory Documents. *LWDA*, pp. 26–37.

[2] M. C. De Marneffe, C. D. Manning, J. Nivre, & D. Zeman, "Universal dependencies," *Computational linguistics*, 47(2), pp. 255–308, 2021. https://doi.org/10.1162/coli_a_00402

[3] K. D'Oosterlinck, S. K. Bitew, B. Papineau, C. Potts, T. Demeester, & C. Develder, CAW-coref: Conjunction-Aware Word-level Coreference Resolution. *arXiv preprint arXiv:2310.06165*. https://doi.org/10.18653/v1/2023.crac-main.2

[4] S. Schuster, & C. D. Manning, "Enhanced English universal dependencies: An improved representation for natural language understanding tasks," *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, 2016, May, pp. 2371–2378.

[5] C. D. Manning, D. Christopher, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP Natural Language Processing Toolkit," *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2014, pp. 55–60. https://doi.org/10.3115/v1/P14-5010

[6] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton and Christopher D. Manning, "Stanza: A Python Natural Language Processing Toolkit for Many Human Languages," *Association for Computational Linguistics (ACL) System Demonstrations*, 2020. https://doi.org/10.18653/v1/2020.acl-demos.14

[7] K. Dong, A. Sun, J. J. Kim, & X. Li, "Syntactic multi-view learning for open information extraction," 2022. *arXiv preprint arXiv:2212.02068*. https://doi.org/10.18653/v1/2022.emnlp-main.272

[8] G. Angeli, M. J. J. Premkumar, & C. D. Manning, "Leveraging linguistic structure for open domain information extraction," *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* (vol. 1: Long Papers), 2015, pp. 344–354. https://doi.org/10.3115/v1/P15-1034

[9] , P. Heyvaert, B. De Meester, A. Dimou, & R. Verborgh, "Declarative rules for linked data generation at your fingertips!," *The Semantic Web: ESWC 2018 Satellite Events: ESWC 2018 Satellite Events*, Heraklion, Crete, Greece, June 3-7, 2018, Revised Selected Papers 15 (pp. 213–217). Springer International Publishing. https://doi.org/10.1007/978-3-319-98192-5_40

[10] D. Van Assche, T. Delva, P. Heyvaert, B. De Meester, & A. Dimou, "Towards a more human-friendly knowledge graph generation & publication," *ISWC2021, The International Semantic Web Conference,* vol. 2980, 2021. CEUR.

[11] Guillaume Plique, "Graphology, a robust and multipurpose Graph object for JavaScript," *Zenodo*, Published December 11, 2021 | Version 0.23.2-lib

[12] D. Li, H. Mei, Y. Shen, S. Su, W. Zhang, J. Wang, M. Zu, & W. Chen, "ECharts: a declarative framework for rapid construction of web-based visualization," *Visual Informatics*, 2(2), 136–146, 2018. https://doi.org/10.1016/j.visinf.2018.04.011

Received April 15, 2025

Michael Zgurovsky. ORCID 0000-0001-5896-7466. Doctor of Technical Sciences. Professor. Scientific Supervisor.
Educational-Scientific Institute for Applied System Analysis, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine.
Education: National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine, (1975).
Research interests: cybernetics, mathematics and system analysis, intelligent big data analysis, decision making theory.
Publications: more than 820 papers.
E-mail: mzz@kpi.ua

Andriy Boldak. ORCID 0000-0001-7151-0743. Candidate of Engineering Sciences. Associate Professor.
Department of Computer Engineering, Faculty of Informatics and Computer Engineering, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine.
Education: National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine, (1987).
Research interests: data reliability, validity and consistency, multivariate data exploration techniques, data integration and indices design, application of cause-effect models, image processing and patterns recognition, design and development information system.
Publications: more than 25 papers.
E-mail: boldak@wdc.org.ua

Kostiantyn Yefremov. ORCID 0000-0003-3495-6417. Candidate of Technical Sciences. Acting Director.
Educational-Scientific Institute for Applied System Analysis, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine.
Education: National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine, (2003).
Research interests: problem-oriented systems based on web-technology, multi-agent systems, data mining, data integration, artificial intelligence, sustainable development.
Publications: more than 50 papers.
E-mail: k.yefremov@wdc.org.ua

Vitalii Statkevych. Candidate of Physico-mathematical Sciences. Researcher.
Department of Mathematical Methods of System Analysis, Educational-Scientific Institute for Applied System Analysis, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine.
Education: National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine, (2011).
Research interests: functional analysis.
Publications: more than 15 papers.
E-mail: mstatkevich@yahoo.com

Pokhylenko Oleksandr. ORCID 0000-0002-1562-2051. Postgraduate Student.
Department of Artificial Intelligence, Educational-Scientific Institute for Applied System Analysis, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine.
Education: National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine, (2023).
Research interests: information extraction, intelligent systems, artificial intelligence.
Publications: more than 10 papers.
E-mail: o.pokhylenko@kpi.ua

М. З. Згуровський, А. О. Болдак, К. В. Єфремов, В. М. Статкевич, О. А. Похиленко. Формальна мова для аналізу графових моделей та її програмна імплементація

Метою статті є розробка спеціалізованої мови для обробки графових даних та її програмної реалізації. Запропоноване рішення забезпечує універсальність, зручність і ефективність, надаючи можливість виконання як базових операцій над графами, так і складніших процедур. Інструмент підтримує класичні алгоритми роботи з графами, зокрема пошук найкоротших шляхів, обхід графів, побудову мінімального кістякового дерева тощо, а також використовується для моделювання та аналізу процесів розповсюдження повідомлень і обробки графових репрезентацій текстових даних. Недоліком багатьох існуючих інструментів для аналізу графів, здебільшого реалізованих як бібліотеки універсальних мов програмування, є їхня незручність у використанні. Це зумовлено тим, що опис процедур для аналізу графів базується на операціях зі структурами даних, визначених у термінах цих універсальних мов, що ускладнює сприйняття та знижує наочність математичних методів, які реалізуються. Розробка спеціалізованої проблемно-орієнтованої мови, побудованої на основі абстракцій високого рівня, дозволить усунути ці недоліки. Така мова забезпечить формалізований опис методів аналізу та обробки графових моделей, підвищуючи їхню зрозумілість і доступність для користувачів. Програмна реалізація цієї мови дозволить отримати готові до використання рішення для виконання методів аналізу графів. Композиція таких методів сприятиме розв'язанню широкого спектра завдань, зокрема аналізу

текстових повідомлень природної мови, а також вивченню процесів оприлюднення та поширення інформації в інтернет-середовищі.

Ключові слова: графові моделі; мова для аналізу графових моделей; семантичні мережі; синтаксичний аналіз; поширення повідомлень; процес поширення чуток.

Згуровський Михайло Захарович. ORCID 0000-0001-5896-7466. Доктор технічних наук. Професор. Науковий керівник.
Навчально-науковий інститут прикладного системного аналізу, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна.
Освіта: Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна, (1975).
Наукові інтереси: кібернетика, математика та системний аналіз, інтелектуальний аналіз великих даних, теорія прийняття рішень.
Публікації: понад 820 статей.
E-mail: mzz@kpi.ua

Болдак Андрій Олександрович. ORCID 0000-0001-7151-0743. Кандидат технічних наук. Доцент.
Кафедра комп'ютерної інженерії, Факультет інформатики та комп'ютерної інженерії, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна.
Освіта: Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна, (1987).
Наукові інтереси: надійність, обґрунтованість і послідовність даних, методи дослідження багатовимірних даних, інтеграція даних і проектування показників, застосування причинно-наслідкових моделей, обробка зображень і розпізнавання образів, інформаційна система проектування і розробки.
Публікації: понад 25 статей.
E-mail: boldak@wdc.org.ua

Єфремов Костянтин Вікторович. ORCID 0000-0003-3495-6417. Кандидат технічних наук. В. о. директора.
Навчально-науковий інститут прикладного системного аналізу, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна.
Освіта: Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна, (2003).
Наукові інтереси: проблемно-орієнтовані системи на основі веб-технологій, мультиагентні системи, добування даних, інтеграція даних, штучний інтелект, сталий розвиток.
Публікації: понад 50 статей.
E-mail: k.yefremov@wdc.org.ua

Статкевич Віталій Михайлович. Кандидат фізико-математичних наук. Науковий співробітник.
Кафедра математичних методів системного аналізу, Навчально-науковий інститут прикладного системного аналізу, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна.
Освіта: Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна, (2011).
Наукові інтереси: функціональний аналіз.
Публікації: понад 15 статей.
E-mail: mstatkevich@yahoo.com

Похиленко Олександр Андрійович. ORCID 0000-0002-1562-2051. Аспірант.
Кафедра штучного інтелекту, Навчально-науковий інститут прикладного системного аналізу, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна.
Освіта: Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна, (2023).
Наукові інтереси: вилучення інформації, інтелектуальні системи, штучний інтелект.
Публікації: понад 10 статей.
E-mail: o.pokhylenko@kpi.ua