Issue 18, part 2, 2025

https://doi.org/10.23939/sisn2025.18.2.096

UDC 004.89

A PATTERN SEARCH ALGORITHM IN GRAPH REPRESENTATIONS OF TEXTUAL DATA FOR AN ONTOLOGY CONSTRUCTION SYSTEM

Andrii Chornyi[1], Andrii Berko[2]

[1, 2] Lviv Polytechnic National University,
Information Systems and Networks Department, Lviv, Ukraine,
[1] E-mail: andrii.o.chornyi@lpnu.ua, ORCID: 0009-0007-4005-4088
[2] E-mail: andrii.y.berko@lpnu.ua, ORCID: 0000-0001-6756-5661

© *Chornyi A., Berko A., 2025*

The article presents the development and formalization of an algorithm for pattern matching in graph representations of textual data as a core component of syntactic-semantic transformations for ontology construction from text documents. The study aims to bridge the gap between natural language processing and formal logic by introducing a universal SPARQL-based approach for executing transformation rules directly on graph database servers. The proposed method integrates RDF data representation with formal graph transformation techniques, including Double Pushout (DPO), ensuring correctness and mathematical rigor. Through the use of graph indexing schemes such as SPO, POS, and OSP, the proposed approach transforms the computationally expensive subgraph isomorphism task from exponential to practical polynomial complexity. The implementation achieves up to 73% runtime reduction during repeated executions due to server-side caching. The research contributes a flexible, formalized, and scalable mechanism for automatic ontology construction, facilitating deep semantic analysis and causal reasoning from textual sources. The algorithm's extensibility allows dynamic rule introduction without recompilation, making it suitable for applications in semantic web systems, knowledge extraction, and AI-driven natural language understanding.

Within the scope of this research, an algorithm was developed and analyzed for identifying homomorphic and isomorphic matches of pattern subgraphs within syntactic graphs, leveraging RDF representations and SPARQL queries enhanced with filter generation algorithms for shape-based matching. The study demonstrates that the complexity of pattern search can be effectively mitigated through graph database indexing strategies, such as SPO, POS, and OSP indexes, reducing exponential complexity to polynomial levels for practical text block sizes. Experimental evaluation confirmed the scalability and efficiency of the proposed approach, revealing substantial runtime reductions during repeated executions as a result of server-side caching.

The work contributes flexible, formalized, and efficient methods for automatic ontology construction from natural language texts, enabling deep semantic analysis and causal reasoning. The approach supports extensibility and dynamic rule introduction without code recompilation, making it suitable for real-world semantic web and knowledge extraction systems. The results have implications for NLP, ontology engineering, and applications requiring interpretability and scalability in processing complex textual data.

Keywords - ontology construction, syntactic-semantic transformation, SPARQL, pattern matching, graph isomorphism, natural language processing, rule formalization.

Problem statement

The search for a compromise solution in representing deductive rules for ontology construction from text is a pressing issue situated at the intersection of natural language processing and logical formalism in ontology engineering. The challenge lies in developing a set of rules that are both intuitively comprehensible to humans and, at the same time, formally rigorous and accessible for automated computer processing. Contemporary approaches demonstrate that successful ontology construction from text is feasible only through the use of logical axioms expressed in a controlled language, which reduces the inherent ambiguity of natural language while preserving sufficient expressiveness for knowledge modeling. Thus, the compromise is achieved by designing formal yet flexible constructs that can subsequently be translated into description logic or related formalisms.

One promising direction involves the development of logical constructs that are independent of any specific programming language or tool, thereby enhancing both the interoperability and the comprehendsibility of the rules for humans and machines alike.

The extension of the deductive approach to the task of natural language text recognition, with consideration of causal reasoning, opens new opportunities in semantic analysis and machine learning. Rule-oriented constructs, in which the left-hand side represents the cause and the right-hand side the effect, enable the direct formalization of causal relations within ontologies. This, in turn, enhances the depth of modeling by allowing not only the extraction of facts but also the interpretation of causal interactions in texts–an aspect critically important for intelligent analysis and decision-making in complex domains. The formalization of deductive rules for ontology construction from text holds the potential to transcend the boundaries of NLP, addressing diverse causality-related tasks across fields ranging from medicine to the social sciences and engineering.

Analysis of Recent Studies and Publications

One of the earliest practical implementations of a rule-oriented approach to ontology construction from text was proposed in the OntoHarvester system (Mousavi, Kerr, Iseli, & Zaniolo, 2014). This system is based on deep NLP and employs the SemScape framework, which transforms text into structured graphs (TextGraphs). In these graphs, the vertices represent candidate terms, while the edges correspond to grammatical relations.

The idea of employing graph databases for representing syntactic graphs with the aim of unifying data from multiple text parsers was introduced in the work of (Chornyi & Dosyn Dmytro, 2025). This approach is intended to improve the accuracy of subsequent ontology construction through rule-oriented transformations into semantic graphs. Furthermore, using a graph database server with the Apache Jena Fuseki Java library, the study demonstrates the possibility of optimizing transformation efficiency by means of query result caching. The findings indicate that repeated execution of data retrieval queries from a syntactic graph is significantly faster than the initial execution. This feature, combined with the formalization of syntactic-semantic transformation rules within an ETL (Extract, Transform, Load) process – where Extract denotes retrieval from the syntactic graph – opens new opportunities for performance optimization through the clustering of queries by dataset. In this work, a practical implementation of syntactic-semantic transformation as an ETL process was developed at the level of a graph database server using the widely adopted SPARQL query language. Such an implementation allows new rules to be introduced on the fly, since executing new scripts does not require code compilation. Nevertheless, challenges remain with the classification and clustering of scripts, as this would necessitate the development of a custom SPARQL interpreter. Consequently, as part of the continued development of the ontology construction system from text, the formalization of transformation rules in a graph-based representation was proposed.

In addition, ETL is traditionally understood as a data integration workflow oriented toward batch processing, in which data are typically transformed outside the target system prior to loading. ETL operates in the context of data streams and practical data handling, with its transformations being less formalized and primarily optimized for the efficient processing of large data volumes. Overall, ETL may constitute a chain of multiple transformations involving different architectural layers.

The formalization of rules for syntactic–semantic transformations at the graph level necessitates the adoption of a more atomic formalism, one that provides mathematically grounded operations with guarantees of correctness. One such formalism is the Double-Pushout (DPO) approach. In particular, (Söldner & Plump, 2024) present a general formalization of the foundations of the DPO method for graph transformations. Their work offers an in-depth and rigorous formalization of DPO graph transformation theory, accompanied by original and detailed proofs of fundamental results. Of particular practical importance for graph-based syntactic–semantic transformations is the validation of the Church–Rosser property, which ensures that parallel and independent rule applications can be reordered while still yielding isomorphic results. To this end, the concepts of parallel and sequential independence were introduced, with the proofs relying on the decomposition and permutation of pushout diagrams.

Authors (Andersen, et al., 2024) present a method for the automatic inference of graph transformation rules, which combines generative and dynamic approaches. The authors posit that the DPO framework serves as the foundational basis for this method. The proposed approach is oriented toward reverse engineering: known system transitions (i.e., graph dynamics) are provided as input, and the task is to construct a minimal set of DPO rules capable of reproducing these transitions. Furthermore, the DPO formalism enables the formalization and resolution of the model compression problem: initially, maximal rules corresponding to each individual transition are reduced to a set of candidate subrules, from which a minimal subset is selected that generates all transformation transitions. This approach is particularly valuable for creating graph dynamic models that are both compact and accurate.

A graph transformation approach similar to DPO, namely the Pushout-Image (POIM) method, is presented by (Duval, Echahed, & Prost, 2020). This approach provides a formal modeling of RDF and SPARQL based on category theory and algebraic graph transformations. RDF graphs and SPARQL templates are treated as objects of specific categories, with a clear distinction between the roles of variables and blank nodes. POIM interprets CONSTRUCT queries as graph rewriting rules $L \rightarrow K \leftarrow R$ and executes them in two steps: first, the input graph is merged with the template by substituting variables and creating new blank nodes; second, all elements not included in the result are pruned. The authors demonstrate that the approach reproduces the official SPARQL semantics for both CONSTRUCT and SELECT queries, with SELECT queries implemented via translation into corresponding CONSTRUCT queries. Unlike classical methods such as DPO or MPOC-PO, POIM is specifically oriented toward generating query results without retaining extraneous data and can be generalized to other graph data models, with the potential for optimization through efficient matching algorithms.

A somewhat simpler approach to graph transformations is the Single Pushout (SPO) method, which is described in (König & Stünkel , 2020) as a categorical approach to graph transformations based on the use of partial morphisms and a single pushout. The core idea is that element deletion is performed automatically together with associated edges, which simplifies rule application and renders this approach technically less restrictive compared to the Double Pushout (DPO) method. The key distinction from DPO lies in the absence of a gluing condition and the need to compute a complement, allowing SPO to apply rules more rapidly, albeit with less stringent control over the structural integrity of the graph.

In another study (Stünkel & König, 2021), SPO is presented as a practically oriented tool for transformations in systems of graph-like structures. The authors emphasize that, due to its simpler application conditions, SPO is well-suited for experimental scenarios, particularly those permitting the automatic removal of “dangling” elements. Unlike DPO, which focuses on rigorous analysis of properties and rule conflicts, SPO provides greater flexibility and speed in syntactic-semantic transformation tasks, although additional mechanisms are required to restore integrity in cases where semantics play a critical role.

In the work (Mežnar, Bevec, Lavrač, & Škrlj, 2022), SPO is discussed within the broader context of graph-based methods applied to ontology learning and completion tasks. The authors highlight that flexible graph transformations, such as SPO, can serve as a mechanism for preprocessing and restructuring knowledge structures, which are subsequently utilized by machine learning algorithms. This facilitates the creation of new connections between concepts and the refinement of semantic relationships, which is

essential for ontology modeling. Unlike DPO, where strictness and conflict control are paramount, SPO enables rapid and intuitive modifications, making it a practical tool for systems focused on integrating syntactic and semantic levels of knowledge processing.

The first and characteristic step common to any approach to graph transformation (SPO, DPO, or POIM) is the search for a subgraph in the source graph according to a given pattern. In the study (Mennicke, Nagel, Kalo, Aumann, & Balke, 2017), the SPARQL query language – designed for working with RDF data – is examined in relation to classical notions of pattern matching in graph theory. The authors demonstrate that basic graph patterns (BGPs) combined with filters in SPARQL can reproduce results equivalent to subgraph isomorphism, whereas the use of BGPs alone without filters corresponds to matches based on dual simulation. They further analyze the limitations of dual simulation and show that the introduction of local constraints on match size and the pruning of irrelevant elements makes it possible to formulate queries that capture strong simulation, which is computationally far more efficient than isomorphism. This work thus provides a novel perspective on the expressive power of SPARQL and outlines the potential of strong simulation for optimizing query processing in large graph databases. A comparative table of the three approaches discussed in the study is provided below.

Table 3

Approaches to Pattern Matching in Graphs (Mennicke, Nagel, Kalo, Aumann, & Balke, 2017)

Approach	Matching Requirements	Match Size	Correspondence in SPARQL	Computational Complexity	Specific Features
Subgraph Isomorphism	The match structure must fully correspond to the pattern (bijective mapping of nodes and edges)	Exactly as in the pattern	BGP + inequality filter on all variables (bijectivity)	NP-complete	The strictest match, with no “extra” nodes or edges
Dual Simulation	Edge direction is preserved both forward and backward; additional nodes/edges are allowed	May be arbitrary	BGP only (without filter)	Polynomial	May return large matches, including irrelevant elements
Strong Simulation	As in dual simulation, but: (1) size is bounded by the pattern diameter, (2) only relevant nodes and edges are retained	Locally bounded by diameter	Combination of BGPs + diameter constraint (via join and center check)	Polynomial	More controlled matching, suitable for search optimization

A significant role in the computational efficiency of each pattern-matching algorithm is played by the technologies of storage, indexing, processing, and optimization of SPARQL queries in graph database management systems.

In particular, the efficiency of storing RDF graphs using different indexing approaches for rapid pattern-based triple retrieval is examined in the study (Ali, Saleem, Yao, Hogan, & Ngonga Ngomo, 2021). The most common approach is the creation of six indexes for all permutations of triples: SPO, SOP, PSO,

POS, OSP, and OPS. These indexes enable efficient support for all possible query patterns, ensuring $O(\log(n))$ access time to data regardless of the position of variables or constants within a triple. Some systems employ three key indexes (SPO, POS, OSP), which already cover all query types, while others construct the full set of six to achieve higher performance. The article also discusses more advanced types of indexes, such as entity-based, property-based, and path indexes, which are used to optimize star queries, path queries, and complex structural searches in graphs.

As graph database management systems continue to evolve rapidly, research and development efforts increasingly focus on novel indexing methods in addition to the standard ones. In particular, the work (Al-Ghezi & Wiese, 2024) demonstrates that for the efficient operation of distributed triple stores, it is not only the presence of basic indexes such as SPO, POS, and OSP that matters, but also the adaptation of their configuration to the current workload. The dynamic index management approach implemented in the UniAdapt system proposed by the authors improves SPARQL query performance while optimizing the use of limited disk space and main memory.

The study (Salehpour & Davis, 2021) introduces another novel approach to RDF data indexing for the optimization of SPARQL query execution, based on the concept of Extended Characteristic Sets (ECS). The primary focus is on how this index enables efficient processing of complex queries involving numerous object-subject and subject-subject joins, which typically pose challenges for traditional systems.

The ongoing trend of continuous development and optimization in graph database management systems suggests the appropriateness of performing syntactic-semantic transformations directly at the database level using the SPARQL language. Moreover, this approach provides flexibility in selecting a graph database management system and identifying the optimal solution for improving the overall performance of syntactic-semantic transformation algorithms.

Formulation of the Article's Objective

The objective of this study is to develop and formalize an algorithm for pattern matching in graph representations of textual data as a key stage of syntactic–semantic transformations. This approach will enable an intuitive, human-understandable interpretation of transformation rules and provide a foundation for a universal mechanism for their subsequent execution at the graph database server level using the SPARQL language.

Main Results

Regardless of the specific approach to syntactic–semantic transformation, each rule typically consists of a left-hand side (LHS), which defines the structure to be matched in the syntactic graph, and a right-hand side (RHS), which specifies the structure to be created in the semantic graph.

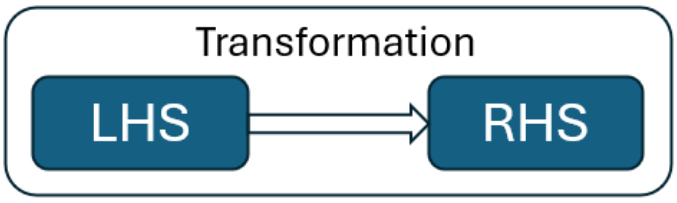

Fig. 1. General scheme of a syntactic–semantic transformation rule

In all cases, the initial stage of the transformation involves a common task – finding a pattern in the source graph, in this instance, in the syntactic graph. The pattern is defined as the subgraph specified in the left-hand side of the transformation rule. Furthermore, the algorithm for solving this task exhibits the highest time complexity compared to the other stages of syntactic–semantic transformation.

The main objective of the study at this stage is to represent the pattern of the target syntactic subgraph in RDF format. To demonstrate the approach, a rule for creating person entities mentioned in the text in the <First Name> – <Last Name> format is used (Chornyi & Dosyn Dmytro, 2025), specifically its left-hand side, which can be described by the following condition:

If there exists a “compound” relation between nodes in the syntactic graph, and the subject and object of this relation are, in turn, connected to the literal “PERSON” via a NamedEntityTagAnnotation relation.

```
@prefix nlp:    <http://nlp.stanford.edu#> .

[] nlp:NamedEntityTagAnnotation "PERSON" ;
   nlp:compound [
     nlp:NamedEntityTagAnnotation "PERSON"
   ] .
```

The first challenge in the task of searching for graph patterns specified in RDF format is the representation of abstract entities, in this case, nodes. In RDF, this is addressed through the concept of a blank node. Thus, the pattern described above can be represented as follows.

The next challenge in addressing this task was the creation of a universal SPARQL script capable of applying an RDF pattern with abstract (blank) nodes to a syntactic graph. To achieve this, the following graph intersection logic was employed.

Let a pattern graph G_P and a syntactic graph G be given, both consisting of triples.

$$\begin{gathered} t = (s,p,o) \mid t \in G_P \\ t' = (s',p',o') \mid t' \in G \end{gathered} \tag{1}$$

Then, the comparison of triples from these two graphs can be expressed as follows.

$$(s,p,o) \equiv (s',p',o') \Leftrightarrow \begin{cases} p = p' \\ (s = s' \vee Blank(s)) \\ (o = o' \vee Blank(o)) \end{cases} \tag{2}$$

Thus, the intersection of a pattern graph with a syntactic graph can be represented as (3).

$$\begin{gathered} G \cap G_P = \{(s',p',o') \in G \mid \exists (s,p,o) \in G_P : (p = p') \wedge \big(s = s' \vee Blank(s)\big) \wedge \\ (o = o' \vee Blank(o))\} \end{gathered} \tag{3}$$

In the form of a SPARQL script, such an intersection can be represented as a transformation into the resulting graph D as illustrated in Code snippet 1, where:

`<http://localhost:3330/opgraph/syngraph>` is the syntactic graph,

`<http://localhost:3330/opgraph/rule1/dpo/l>` is the pattern graph,

`<http://localhost:3330/opgraph/morphism/dpo/d>` is the resulting graph.

A more complex task is the identification of abstract relations, since these are not predefined in RDF. However, this can be addressed by representing the relation as a constant within a custom namespace, serving as an artificial auxiliary entity, for example, provisionally as `<crocus:abstractRelation>`.

However, such a solution would significantly reduce the level of abstraction in the graph-based representation of patterns, tying them to a specific implementation through a particular namespace. To avoid this, the parent class of all predicates in the RDF namespace, namely <rdf:Property> (RDF 1.2 Schema, 2025), was used to represent abstract relations (predicates) in the patterns of the syntactic-semantic transformation.

```
INSERT
{
      GRAPH <http://localhost:3330/opgraph/morphism/dpo/d>
      {
            ?sd ?p ?od.
      }
}
WHERE
{
      GRAPH <http://localhost:3330/opgraph/syngraph>
      {
            ?sg ?p ?og.
            BIND(?sg AS ?sd)
            BIND(?og AS ?od)
      }
      GRAPH <http://localhost:3330/opgraph/rule1/dpo/l>
      {
            ?s ?p ?o.
            BIND(IF(isBLANK(?s), ?sg), ?s) AS ?sd)
            BIND(IF(isBLANK(?o), ?og), ?o) AS ?od)
      }
}
```

Code snippet 1. Intersection of the graph G and the pattern G_P with abstract nodes in SPARQL.

This formalism falls within the generally accepted concept according to which each triple contains a predicate, and every predicate belongs to the class `rdf:Property`.

$$\forall s \forall p \forall o(Triple(s,p,o) \rightarrow Property(p))$$
$$Property(p) \equiv type(p, rdf{:}Property) \tag{4}$$

Accordingly, Code snippet 1 was transformed into the Code snippet 2.

Since this script compares each triple individually, it does not perform the task of subgraph pattern matching but rather executes a “content-based” selection, instantiating the abstraction. In other words, it searches for homomorphic matches of the pattern G_P within the graph G.

Considering that the most computationally intensive operation in this script is the joining of triples from both graphs, its complexity was evaluated as follows.

Let n_G denote the number of triples in the syntactic graph, and n_P the number of triples in the pattern graph. Then, the complexity of triple matching for these graphs is $O(n_G)$ and $O(n_P)$, respectively.

Thus, the worst case corresponds to a naïve join with a complexity of $O(n_G \times n_P)$.

In practice, the use of indexes such as SPO, POS, OSP, and others significantly reduces the extent of graph traversal. Without indexes, the execution time grows exponentially, whereas with indexes it approaches near-linear behavior even for large graphs (up to 100,000 nodes), as demonstrated in the study (Pokorný, Valenta, & Troup, 2018). The complexity of this operation in a real-world setting is estimated to be (5).

$$O(\min(n_G, n_P) \times \log(\max(n_G, n_P))) \tag{5}$$

Since in our case the pattern graph is always significantly smaller than the syntactic graph $n_P \ll n_G$, the actual (practical) complexity of this script can be expressed as (6).

$$O(n_P \times \log(n_G)) \tag{6}$$

Thus, the script presented in Code snippet 2 plays a crucial role in the graph pattern search algorithm. First, it is universal for all types of RDF patterns and therefore does not require additional logic at the application level of syntactic-semantic transformation. Second, it generates a graph for further processing

that is smaller than the input syntactic graph, thereby reducing the computational cost of subsequent operations. Third, at this stage, an important task is addressed – the instantiation of abstract nodes and predicates, which in many cases constitute an integral part of graph patterns. Fourth, taking data indexing into account, the algorithm exhibits relatively low complexity.

```
PREFIX rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#

INSERT
{
      GRAPH <http://localhost:3330/opgraph/morphism/dpo/d>
      {
            ?sd ?pd ?od.
      }
}
WHERE
{
      GRAPH <http://localhost:3330/opgraph/syngraph>
      {
            ?sg ?pq ?og.
            BIND(?sg AS ?sd)
            BIND(?og AS ?od)
      }
      GRAPH <http://localhost:3330/opgraph/rule1/dpo/l>
      {
            ?s ?p ?o.
            BIND(IF(?p = rdf:Property, ?pg, ?p) AS ?pd)
            BIND(IF(isBLANK(?s), ?sg), ?s) AS ?sd)
            BIND(IF(isBLANK(?o), ?og), ?o) AS ?od)
      }
}
```

Code snippet 2. Intersection of the graph G and the pattern G_P with abstract nodes and predicates in SPARQL.

The result of the intersection $G \cap G_P$ using the script from Code snippet 2 is a set of triples.

$$T_d = \{t \in G \mid Match(t, G_P)\} \tag{7}$$

That is, the set T_d contains all triples of the graph G that match at least one triple of the pattern graph G_P, but are not necessarily part of the subgraph D_i corresponding to the pattern G_P.

The next challenge in this study was the search for isomorphic matches with the pattern graph G_P within the set T_d. In other words, it was necessary to employ the SPARQL language to reconstruct the template in the form shown in Fig. 2, which is not explicitly evident in $G \cap G_P$.

As shown in Fig. 2, shaping the set of three triples requires the imposition of certain conditions, namely, the equality of nodes for forming a star or a chain, as well as the equality of edges.

To address the task of searching for isomorphic matches with the pattern graph depicted in Fig. 2, a SPARQL script (Code snippet 3) was developed. In the INSERT clause, it constructs a meta graph which contains subgraphs representing the identified isomorphic correspondences to G_P. The actual search for isomorphisms is performed within the WHERE clause of the script. Isomorphism is achieved through the use of the bijection $\mathbb{R}^{n_P \times n_P} \leftrightarrow \mathbb{R}^{n_P^2}$ and the application of the aforementioned conditions by means of SPARQL filters (Mennicke, Nagel, Kalo, Aumann, & Balke, 2017).

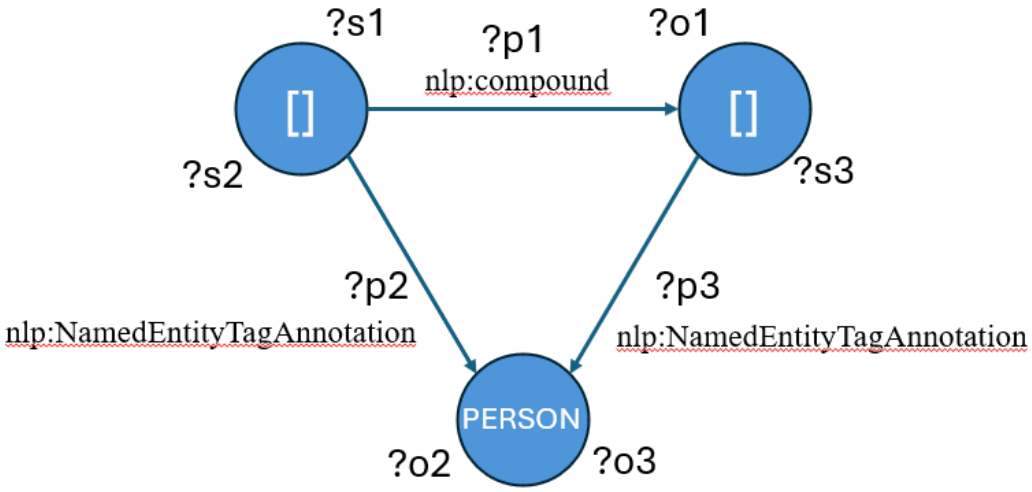

Fig. 2. A pattern graph for person extraction from text

The bijection $\mathbb{R}^{n_P \times n_P} \leftrightarrow \mathbb{R}^{n_P^2}$, in this case, ensures the selection of a set of triples corresponding to the size of the pattern graph at each stage. This is achieved through vectorization, where the vector represents a set of triples $T \in T_d$ of size $|G_P|$, with G_P being the pattern graph. Accordingly, considering that the set of triples T forms a matrix with three columns (s, p, o), the bijective function for this case takes the following form.

$$\mathbb{R}^{n_P \times 3} \leftrightarrow \mathbb{R}^{3n_P}, \tag{8}$$

where $n_P = |G_P|$.

In turn, the application of filters to the vector $\mathbb{R}^{3n_P}$ ensures the verification of conformity with the pattern's structure.

$$f: \mathbb{R}^{3n_P} \to \{true, false\} \tag{9}$$

The algorithmic complexity of the script presented in Code snippet 3 can be formally expressed as follows.

$$O = O_I + O_W\,, \tag{10}$$

where O_I – the complexity of INSERT clause, O_W – the complexity of WHERE clause.

The complexity of the INSERT clause is always proportional to the amount of data, which in this case is the product of the number of identified matches and the size of the pattern graph G_P.

$$O_I = O(kn_P)\,, \tag{11}$$

where k is the number of identified subgraphs D_i corresponding to the pattern G_P, and $n_P = |G_P|$.

The complexity of the WHERE clause, in the worst-case scenario where database indexing is not considered, is NP-complete with respect to the size of the pattern; that is, if the pattern size is treated as a variable component. Since the set of patterns is finite, the set of permissible $n_P = |G_P|$ is also finite.

$$|G_P| \ll \infty \tag{12}$$

Therefore, it is reasonable to assess the algorithm's complexity for each individual pattern G_P and for variable input data in the form of a graph G.

For convenience, we introduced the coefficient m as the cardinality of the pattern graph.

$$m = n_P = |G_P| \tag{13}$$

In this case, the size of the graph G can be represented by the variable $n = n_G$. Accordingly, the naïve complexity of the WHERE clause can be expressed as (14).

$$O_W = O(n^m) = O(n^{|G_P|}) \tag{14}$$

```
PREFIX : <http://crocus.science/dpo/rule1/l#>

                        INSERT
                          {
                     GRAPH ?dGraph
                          {
                     ?s1 ?p1 ?o1 .
                     ?s2 ?p2 ?o2 .
                     ?s3 ?p3 ?o3 .
                          }
       GRAPH <http://localhost:3330/opgraph/meta>
                          {
                ?dGraph :type :subgraph
                          }
                          }
                        WHERE
                          {
    SELECT ?dGraph ?s1 ?p1 ?o1 ?s2 ?p2 ?o2 ?s3 ?p3 ?o3
                        WHERE
                          {
   GRAPH <http://localhost:3330/opgraph/morphism/dpo/d>
                          {
                      ?s1 ?p1 ?o1.
                      ?s2 ?p2 ?o2.
                      ?s3 ?p3 ?o3.
                          }
                   FILTER (?s3 = ?o1)
                        FILTER (?s2 = ?s1)
                  FILTER (?p2 != ?p1)
                       FILTER (?p3 != ?p1)

BIND(IRI(CONCAT("http://localhost:3330/opgraph/d-graph-",
                      STRUUID())) AS ?dGraph)
                          }
                          }
```

Code snippet 3. Searching for isomorphic matches with the pattern G_P using SPARQL.

Taking graph database indexing into account, specifically the use of SPO, POS, and OSP indexes, the complexity degree decreases by at least one order (15).

$$O_W = O(n^{m-1}) \tag{15}$$

Thus, the overall complexity of the script presented in Code snippet 3, for each pattern with cardinality m, can be expressed as a polynomial of degree $m - 1$.

$$O = O(n^{m-1}) + O(km)\,, \tag{16}$$

where n is the size of the input graph, represented by the set T_d generated by executing the script shown in Code snippet 2; m is the cardinality of the pattern graph G_P relative to which the transformation is performed; and k is the number of identified subgraphs D_i corresponding to the pattern G_P.

Taking these substitutions into account, the overall complexity of the pattern search algorithm, which consists of the sequential execution of the scripts presented in the Code snippet 2 and the Code snippet 3, respectively, takes the following form.

$$O = O(n^{m-1}) + O(km) + O(m \times \log(n_G)) \tag{17}$$

Let us consider the case where the input graph G consists solely of subgraphs corresponding to the pattern G_P. Then, the number of identified subgraphs D_i will be $k \approx \frac{n}{m}$, which approximates the worst-case scenario. In this case, the following statement (18) holds true.

$$O(km) \sim O(n) \tag{18}$$

In this case, the overall complexity of the pattern search algorithm turns into equation (19).

$$O = O(n^{m-1}) + O(n) + O(m \times \log(n_G)) \tag{19}$$

Unlike the script presented in the Code snippet 2, which generates a homomorphic projection of the pattern G_P onto the graph G and is universal for any patterns and graphs, Code snippet 3 is specific to each pattern G_P, necessitating the implementation of an algorithm for the automated generation of pattern search scripts. To this end, a formalism based on SPARQL filter blocks was introduced, representing filtering operators in pairwise form, as opposed to a set of logical conditions combined using 'AND' or 'OR' operators (Mennicke, Nagel, Kalo, Aumann, & Balke, 2017). This representation provides better insight into regularities and enables the algorithmic generation of a filter set based on the structure of the pattern graph.

In order to develop an algorithm for the automatic generation of SPARQL filters, which are intended to perform 'shape-based' matching with a pattern graph, the basic figures that constitute the overall structure of an RDF graph were formalized (Fig. 3).

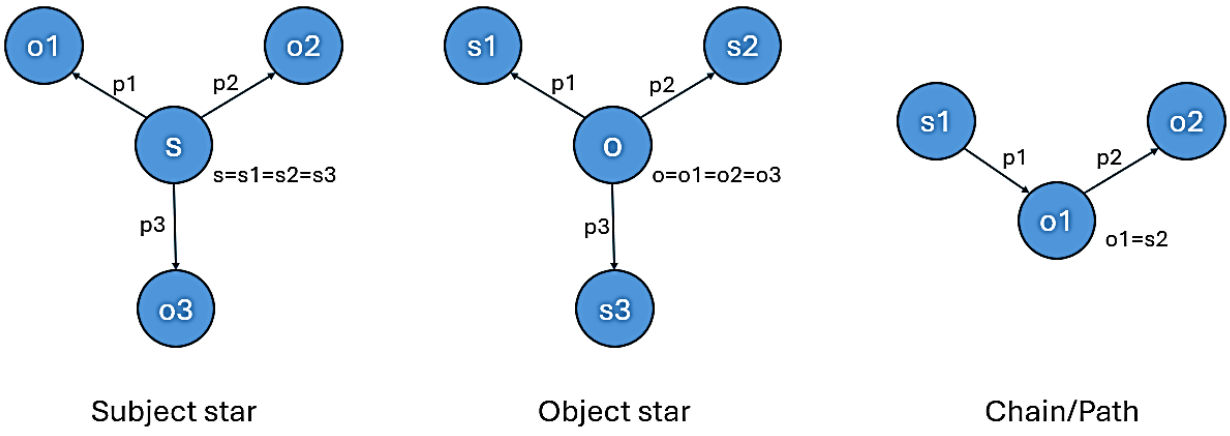

Fig. 3. Basic structures of an RDF graph.

Next, the mathematical model of this algorithm is presented. Let a pattern graph be given.

$$G_P = (T, S, P, O), \tag{20}$$

where $T = \{t_0, t_1, t_2, \dots t_{n-1}\}$ is a set of triples with cardinality $m = |G_P|$, defining the number of triples $t_i = (s_i, p_i, o_i)$, such that $s_i \in S$ (the set of subjects), $p_i \in P$ (the set of predicates/relations), and $o_i \in O$ (the set of objects).

The task was to develop an algorithm that generates a conjunction of constraints in the form of SPARQL filters (21)

$$\Phi(G_P) = \bigwedge FILTER(\cdot) \tag{21}$$

where each filter represents either an equality (=) or inequality (≠) relation between the variables $\{?s_i, ?p_i, ?o_i\}$.

The condition for an isomorphic match between the resulting subgraph D_i and the pattern graph G_P is defined as the correspondence of all basic structures (Fig. 3) composing both D_i and G_P. The filter generation algorithm is implemented via cyclic pairwise comparison of variables within the sets S, P, O and cross-comparison of elements in the sets S, O. To optimize subsequent filter application, exclusion sets are created, which include one of the indices from each pairwise comparison when the equality condition $x_i = x_j$ holds, since this equality eliminates the need to compare other elements with x_j if a comparison with x_i has already been performed.

Thus, the creation of filters for verifying the correspondence of D_i to all subject stars of G_P (Fig. 3) is implemented according to the following logic.

$$\begin{gathered}\forall i, j (0 \le i < j < m)\ \phi_{i,j}^{(s)}: \\ \left[(s_i = s_j) \leftrightarrow (?s_i = ?s_j) \wedge ((s_i = s_j) \rightarrow (j \in E_s))\right]\end{gathered} \tag{22}$$

Similarly, for filters corresponding to object stars (Fig. 3) is expressed as (23).

$$\begin{gathered}\forall i, j (0 \le i < j < m)\ \phi_{i,j}^{(o)}: \\ \left[(o_i = o_j) \leftrightarrow (?o_i = ?o_j) \wedge ((o_i = o_j) \rightarrow (j \in E_o))\right]\end{gathered} \tag{23}$$

The generation of filters to verify the correspondence of D_i to all chains in G_P (Fig. 3) is performed using a cross-comparison logic between the elements of S, O.

$$\begin{gathered}\forall i, j (0 \le i < j < m)\ \ \phi_{i,j}^{(so)}: \\ \left[(s_i = o_j) \leftrightarrow (?s_i = ?o_j) \wedge ((s_i = o_j) \rightarrow ((i, j) \in E_{so}))\right]\end{gathered} \tag{24}$$

An integral part of the isomorphism of the basic graph structures (Fig. 3) is the correspondence of all predicates within them, which is implemented as (25).

$$\begin{gathered}\forall i, j (0 \le i < j < m)\ \phi_{i,j}^{(p)}: \\ \left[(p_i = p_j) \leftrightarrow (?p_i = ?p_j) \wedge ((p_i = p_j) \rightarrow (j \in E_p))\right]\end{gathered} \tag{25}$$

The indices of elements that should not participate in filtering are added to the exclusion sets.

$$E_s \subseteq \{0, \dots, m-1\}, \tag{26}$$

$$E_o \subseteq \{0, \dots, m-1\},$$

$$E_p \subseteq \{0, \dots, m-1\},$$

$$E_{so} \subseteq \{(i,j) | i < j, i < m\}$$

Thus, the resulting constraint function takes the form shown below.

$$\Phi(P) = \bigwedge_{\substack{i<j \\ j \notin E_s}} \phi_{i,j}^{(s)} \wedge \bigwedge_{\substack{i<j \\ j \notin E_p}} \phi_{i,j}^{(p)} \wedge \bigwedge_{\substack{i<j \\ j \notin E_o}} \phi_{i,j}^{(o)} \wedge \bigwedge_{\substack{i<j \\ (i,j) \notin E_{so}}} \phi_{i,j}^{(so)}. \quad (27)$$

The implementation of this algorithm was carried out in Java. By integrating this method into the algorithm for generating the selection vector $\mathbb{R}^{n_P^2}$ of the bijection $\mathbb{R}^{n_P \times n_P} \leftrightarrow \mathbb{R}^{n_P^2}$ and applying it to our pattern graph (Fig. 2), we obtained the SPARQL query shown in the Code snippet 4.

Evidently, the variable indices in the generated SPARQL query (Code snippet 4) depend on the order of the triples in the pattern as read from the graph database. However, comparing Code snippet 4 and Code snippet 3 in terms of filtering, it is clear that the automatically generated filter block is more comprehensive and accounts for constraints that may not be immediately obvious to a human. This implementation ensures the isomorphism of subgraphs $D_i \subseteq G$ with the pattern graph G_P without human intervention.

```
SELECT ?s0 ?p0 ?o0 ?s1 ?p1 ?o1 ?s2 ?p2 ?o2
WHERE
                                    {
      GRAPH <http://localhost:3330/opgraph/rule1/dpo/l>
      {
            ?s0 ?p0 ?o0 .
            ?s1 ?p1 ?o1 .
            ?s2 ?p2 ?o2 .
      }
                          FILTER(?s0 != ?s1)
      FILTER(?p0 != ?p1)
      FILTER(?o0 != ?o1)
      FILTER(?s0 != ?s2)
      FILTER(?p0 = ?p2)
      FILTER(?o0 = ?o2)
      FILTER(?s1 = ?s2)
      FILTER(?s0 = ?o1)
}
```

Code snippet 4. SPARQL query for searching the vector $\mathbb{R}^{n_P^2}$.

The development of an algorithm for the automatic search of morphisms to a pattern graph specified in RDF format enabled the experimental evaluation of its runtime complexity. The algorithm was implemented using the intelligent agent CROCUS (Chornyi & Dosyn Dmytro, 2025). The test data consisted of sets of sentences, each containing a match to the pattern shown in Fig. 2. The number of textual blocks, N_{SENT}, ranged from 1 to 100 sentences, with a step of 1 sentence for the range 1 – 20 and a step of 5 sentences for the range 20 – 100. Additionally, for each block of sentences, the values of n – the cardinality of the triple set T_d generated in the first part of the algorithm (Code snippet 2) – and n_G – the size of the syntactic graph (number of triples) – were computed.

During the study, the actual runtime T of the pattern search algorithm in the syntactic graph was measured, along with the runtime upon repeated execution T_{rep} (to evaluate the impact of graph database server caching algorithms). Additionally, the reduction in runtime upon repetition, $\Delta T/T$, was calculated, as well as the expected runtime complexity of the algorithm, O_T, based on the previously derived polynomial complexity $O_T = n^{m-1} + n + m \times \log(n_G)$.

The results of the study are illustrated in Fig. 4, Fig. 5 and Fig. 6.

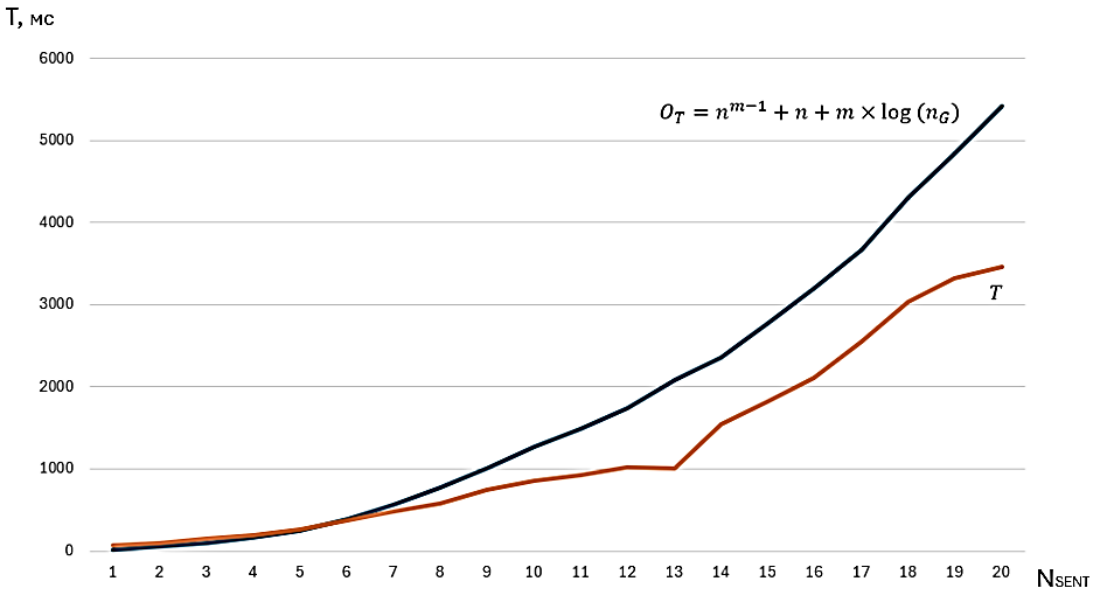

Fig. 4. Calculated and experimental runtime complexity of the algorithm for searching subgraphs isomorphic to a pattern of cardinality m=3 in the syntactic graph of a text block ranging from 1 to 20 sentences.

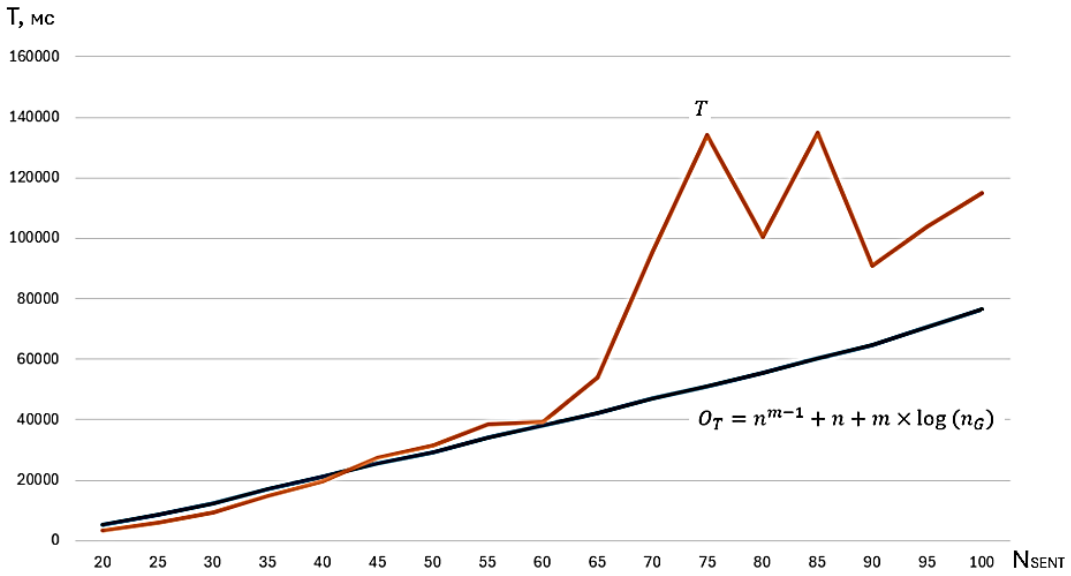

Fig. 5. Calculated and experimental runtime complexity of the algorithm for searching subgraphs isomorphic to a pattern of cardinality m=3 in the syntactic graph of a text block ranging from 20 to 100 sentences.

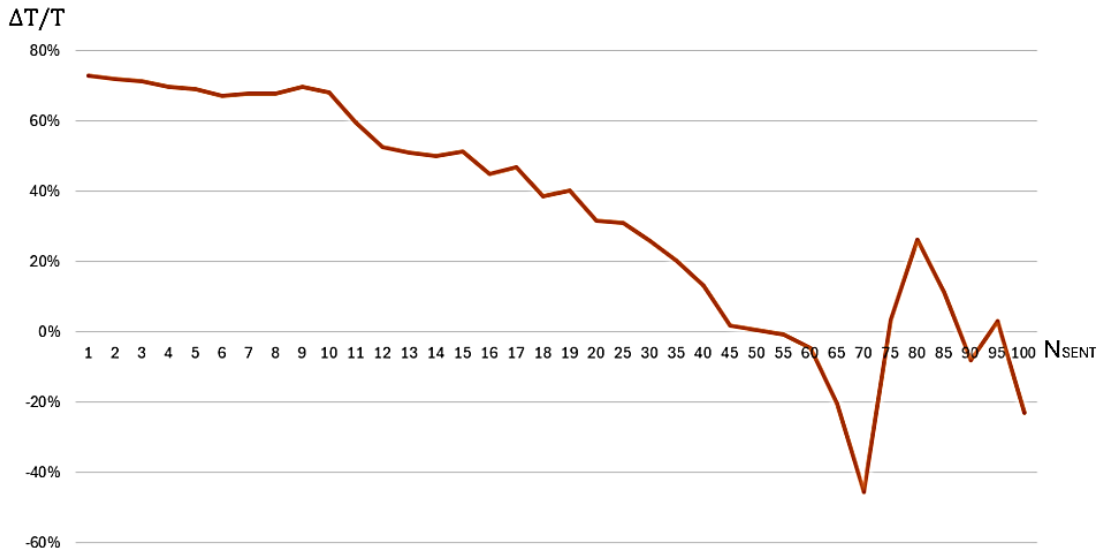

Fig. 6. Relative runtime reduction of the algorithm for searching subgraphs isomorphic to a pattern of cardinality m=3 in the syntactic graph of a text block ranging from 1 to 100 sentences upon repeated execution.

Conclusions

The experimental studies conducted in this work demonstrate the temporal complexity of a pattern search algorithm (subgraph isomorphism) within syntactic graphs of textual blocks of varying sizes. The main results indicate the effectiveness of the proposed approach, which leverages a graph database server. Even the use of basic indexing schemes, such as SPO, POS, and OSP, significantly optimizes subgraph search and reduces the theoretically exponential complexity to a practically manageable level.

The results show a stable increase in algorithm execution time as the size of textual blocks grows, consistent with the underlying mathematical complexity model. At the same time, repeated executions of the algorithm are substantially faster due to server-level caching, with observed reductions in execution time of up to 73% for small textual blocks.

It is observed that the algorithm's complexity scales approximately as $O(n^{m-1})$, as the highest degree of equation (19), where n is the size of the input syntactic graph and m is the cardinality of the pattern graph. This finding aligns with theoretical predictions and demonstrates an acceptable level of scalability for practical applications in ontology construction from text.

Fig. 4 illustrates deviations of actual execution time from theoretical expectations when the size of textual blocks exceeds 60 sentences. This behavior is attributed to server cache overflow, as evidenced by the relative execution time reduction observed during repeated runs (Fig. 6). These deviations are not critical, remaining within the bounds of polynomial complexity, but should be considered in practical implementations of text-based ontology construction systems.

Based on the experimental data and analysis, it can be concluded that employing syntactic-semantic transformations with formalized pattern search via SPARQL-based algorithms constitutes an effective approach for the automatic derivation of ontological modeling rules from natural language texts. The results confirm the feasibility of implementing such transformations within graph databases for textual data volumes sufficient to carry contextual information.

This opens prospects for applying the proposed approach to broader semantic analysis tasks, knowledge integration, and natural language understanding, particularly in educational and medical domains where accuracy and scalability of information processing are critical.

Thus, this work makes a significant contribution to the fields of natural language processing and ontology engineering by proposing flexible, formalized, and efficient methods for transforming textual data into semantic graphs, with mathematically guaranteed correctness and practical implementation using graph database servers.

REFERENCES

Al-Ghezi, A., & Wiese, L. (2024). Analyzing workload trends for boosting triple stores performance. *Elsevier Ltd*. doi:doi.org/10.1016/j.is.2024.102420

Ali, W., Saleem, M., Yao, B., Hogan, A., & Ngonga Ngomo, A.-C. (2021). A Survey of RDF Stores & SPARQL Engines for Querying Knowledge Graphs. *The VLDB Journal*, 1-26. doi:doi.org/10.1007/s00778-021-00711-3

Andersen, J. L., Davoodi, A., Fagerberg, R., Flamm, C., Fontana, W., Kolčák, J., . . . Nøjgaard, N. (2024, Apr 3). *Automated Inference of Graph Transformation Rules.* doi:10.48550/arXiv.2404.02692

Chornyi, A., & Dosyn Dmytro. (2025). Development of a unified output format for text parsers in the ontology construction system from text documents. *Journal of Lviv Polytechnic National University "Information Systems and Networks"*. doi:10.23939/sisn2025.17.170

Duval, D., Echahed, R., & Prost, F. (2020). An Algebraic Graph Transformation Approach for RDF and SPARQL. *Eleventh International Workshop on Graph Computation Models (GCM 2020)* (pp. 55-70). EPTCS. doi:10.4204/EPTCS.330.4

König, H., & Stünkel , P. (2020). Single Pushout Rewriting in Comprehensive Systems. *Graph Transformation. ICGT 2020. Lecture Notes in Computer Science(), vol 12150. Springer*, (pp. 91-108). doi:doi.org/10.1007/978-3-030-51372-6_6

Mennicke, S., Nagel, D., Kalo, J.-C., Aumann, N., & Balke, W.-T. (2017). Reconstructing Graph Pattern Matches Using SPARQL. *Lernen, Wissen, Daten, Analysen, LWDA 2017 - Conference Proceedings* (pp. 152-164). Rostock, Germany: CEUR-WS. Retrieved from https://ceur-ws.org/Vol-1917/paper24.pdf

Mežnar, S., Bevec, M., Lavrač, N., & Škrlj, B. (2022). Ontology Completion with Graph-Based Machine Learning: A Comprehensive Evaluation. *Machine Learning and Knowledge Extraction*, 1107-1123. doi:doi.org/10.3390/make4040056

Mousavi, H., Kerr, D., Iseli, M., & Zaniolo, C. (2014). Harvesting Domain Specific Ontologies from Text. *International Conference on Semantic Computing.* Newport Beach, CA, USA. doi:10.1109/ICSC.2014.12

Pokorný, J., Valenta, M., & Troup, M. (2018). Indexing Patterns in Graph Databases. *Proceedings of the 7th International Conference on Data Science, Technology and Applications (DATA 2018)* (pp. 313-321). Science and Technology Publications, Lda. doi: 10.5220/0006826903130321

RDF 1.2 Schema. (2025, September). Retrieved from www.w3.org: https://www.w3.org/TR/rdf12-schema/

Salehpour, M., & Davis, J. G. (2021). A Comparative Analysis of Knowledge Graph Query Performance. *Third International Conference on Transdisciplinary AI (TransAI)*, (pp. 33-40). doi:10.1109/TransAI51903.2021.00014

Söldner, R., & Plump, D. (2024, October 04). Formalising the double-pushout approach to graph transformation. *Logical Methods in Computer Science*, 3:1–3:37. doi:10.46298/LMCS-20(4:3)2024

Stünkel, P., & König, H. (2021). Single pushout rewriting in comprehensive systems of graph-like structures. *Theoretical Computer Science*, 23-43. doi:doi.org/10.1016/j.tcs.2021.07.002

АЛГОРИТМ ПОШУКУ ШАБЛОНІВ У ГРАФОВОМУ ПОДАННІ ТЕКСТОВИХ ДАНИХ ДЛЯ СИСТЕМИ ПОБУДОВИ ОНТОЛОГІЇ

Андрій Чорний[1], Андрій Берко[2]

[1, 2] Національний університет "Львівська політехніка",
кафедра інформаційних систем та мереж, Львів, Україна
[1] E-mail: andrii.o.chornyi@lpnu.ua, ORCID: 0009-0007-4005-4088
[2] E-mail: andrii.y.berko@lpnu.ua, ORCID: 0000-0001-6756-5661

© *Чорний А., Берко А., 2025*

У статті представлено розроблення та формалізацію алгоритму пошуку шаблонів у графових представленнях текстових даних як ключового компонента синтаксико-семантичних трансформацій для побудови онтологій із текстових документів. Розглянуто проблему поєднання опрацювання природної мови та логічного формалізму шляхом запропонування універсального механізму на основі SPARQL для виконання правил трансформації на серверах графових баз даних. Підхід використовує графові бази даних для представлення синтаксичних графів та застосовує формальні методи трансформації графів, включаючи метод Double Pushout (DPO), щоб забезпечити математично обґрунтований та коректний пошук шаблонів і застосування правил.

У межах цього дослідження було розроблено та проаналізовано алгоритм для визначення гомоморфних та ізоморфних збігів підграфів шаблонів у синтаксичних графах, використовуючи представлення RDF та SPARQL-запити, доповнені алгоритмами генерації фільтрів для пошуку за формою. Показано, що складність пошуку шаблонів може бути ефективно знижена завдяки стратегіям індексації в графових базах даних, таким як SPO, POS та OSP, що зменшує експоненційну складність до поліноміальної для практичних розмірів текстових блоків. Експериментальна оцінка підтвердила масштабованість та ефективність запропонованого підходу, демонструючи суттєве скорочення часу виконання при повторних запусках завдяки кешуванню на сервері.

Робота вносить гнучкі, формалізовані та ефективні методи для автоматичного побудови онтологій з текстів природною мовою, забезпечуючи глибокий семантичний аналіз та причинно-наслідкове мислення. Підхід підтримує розширюваність та динамічне введення правил без перекомпіляції коду, що робить його придатним для реальних систем семантичних мереж та вилучення знань. Отримані результати мають значення для обробки природної мови, інженерії онтологій та застосувань, що потребують інтерпретованості та масштабованості при обробці складних текстових даних.

Ключові слова – побудова онтологій, синтаксико-семантична трансформація, SPARQL, пошук шаблонів, ізоморфізм графів, обробка природної мови, формалізація правил.