A Multilayer Approach to Subgraph Matching in HP-graphs

Nikolai M. Suvorov
Department of Business Informatics
National Research University
Higher School of Economics
Perm, Russian Federation
E-mail: SuvorovNM@gmail.com

Lyudmila N. Lyadova
Department of Business Informatics
National Research University
Higher School of Economics
Perm, Russian Federation
E-mail: LNLyadova@gmail.com

***Abstract.* Visual modeling is widely used nowadays, but the existing modeling platforms cannot meet all the user requirements. Visual languages are usually based on graph models, but the graph types used have significant restrictions. A new graph model, called *HP*-graph, whose main element is a set of poles, the subsets of which are combined into vertices and edges, has been previously presented to solve the problem of insufficient expressiveness of the existing graph models. Transformations and many other operations on visual models face a problem of subgraph matching, which slows down their execution. A multilayer approach to subgraph matching can be a solution for this problem if a modeling system is based on the *HP*-graph. In this case, the search is started on the higher level of the graph model, where vertices and hyperedges are compared without revealing their structures, and only when a candidate is found, it moves to the level of poles, where the comparison of the decomposed structures is performed. The description of the idea of the multilayer approach is given. A backtracking algorithm based on this approach is presented. The Ullmann algorithm and VF2 are adapted to this approach and are analyzed for complexity. The proposed approach incrementally decreases the search field of the backtracking algorithm and helps to decrease its overall complexity. The paper proves that the existing subgraph matching algorithms except ones that modify a graph pattern can be successfully adapted to the proposed approach.**

Keywords: DSM platform; visual model; subgraph matching; isomorphism; graph model; HP-graph; algorithms on graphs.

I. Introduction

The study of any objects and processes, as well as their design, can barely be done without modeling; that is why software tools that allow specialists to build various models and formalize descriptions of objects and processes, or use modeling as a method of analysis, are becoming more popular. Models are described and built with the help of a visual modeling language, which is a fixed set of graphical symbols and rules for constructing visual models by using these symbols [1]. Visual languages can be represented as various types of graphs, including oriented graphs [2], hypergraphs [3], hi-graphs [4], meta-graphs [5] and *P*-graphs [6].

Previously, a new graph model, called *HP*-graph, was proposed as a formalism for representing visual languages [7]. This model unites expressive possibilities of all the mentioned graph types and, thus, it can be used for building more complicated models than those which can be built with the help of the other graph models. The paper [7] proved that this graph model allows the creation of a flexible visual model editor based on it.

This model is proposed as a basis for domain-specific modeling, one of the key aspects of which is model transformations. Such transformations allow users to move from one level of abstraction to another (a vertical transformation) or from one modeling language to another (a horizontal transformation) [5]. Different approaches can be used to transform visual models, but the current standard is the algebraic approach which is based on the graph grammars [9]. Based on this approach, a transformation $r = (L, R)$ includes the left and the right part, where L is a subgraph to be found in a source graph, and R is a subgraph replacing L in the source graph.

As for the *HP*-graph, only main operations, including operations of adding and removing graph elements and operations of decomposition, were described for this model, and no algorithm were proposed to perform an isomorphic subgraph search operation. The structural complexity of the model requires modifying the existing algorithms to adapt them to this model. The *HP*-graph has a multilayer structure which consists of the layer of vertices and hyperedges and the layer of poles and links, sets of which are combined into the elements of the former layer. The multilayer structure of the graph model allows to reduce time complexity of search algorithms. The number of operations can be decreased due to the fact that the first search and matching is performed on the layer of vertices and hyperedges, and only after finding a subgraph with the desired characteristics, the algorithm moves to a more detailed level, where the already selected sets of corresponding poles and ordinary edges are compared.

In practice, a task of finding an isomorphic subgraph has a wide range of applications, including chemical compound search [10], social network analysis [11], pattern recognition [12], and protein interaction analysis [13]. However, subgraph matching is a bottleneck in the overall performance for most of these applications due to the fact that this task is *NP*-hard [14]. For instance, nodes count for protein structure analysis can reach up to tens of thousands [15]; that is why active efforts are currently being made to find an optimal algorithm for subgraph matching.

In visual modeling the problem is the same. The thesis [5] proposes to represent all the models in the form of a single graph, which allows users to maintain links between the models and automatically propagate changes from the source model to the target ones associated with it. For instance, a change in the metamodel of the subject area should be propagated to all the models built on this metamodel. However, storing all the models as a single graph increases the computational complexity of the algorithms on this graph, which requires developing an efficient subgraph search algorithm for the graph model used.

The contributions of these paper are:

1) a new multilayer approach to decrease complexity of subgraph matching algorithms,
2) a backtracking algorithm based on this approach,
3) applications of this approach in several existing subgraph matching algorithms.

The paper is organized as follows. Section II discusses related work and the main algorithms for finding subgraph isomorphism. Section III presents the proposed graph model, definitions of the *HP*-subgraph and isomorphism of the *HP*-graphs, and the multilayer approach to subgraph matching. Section IV introduces a backtracking algorithm based on this approach. Section V presents several applications of the approach in the existing subgraph matching algorithms. Section VI describes the obtained results. Section VII concludes the paper.

II. Related Work

The problem of subgraph matching has been investigated for many years. The works of many scientists, such as [16]-[18], are dedicated to exploring applicability, time complexity and limitations of the existing subgraph matching algorithms. These algorithms are generally divided into two classes:

- Algorithms that observe many graphs $\{G_1, \ldots, G_n\}$ and retrieve those which contain a query graph Q.
- Algorithms that observe a single graph G and retrieve all its subgraphs which are isomorphic to a query graph Q.

In both of these approaches, algorithms can either return a correct and complete answer (having an exponential time complexity) or return an approximate answer (having a polynomial time complexity). While the complete answers describe all subgraphs exactly isomorphic to a pattern, the approximate answers are generally obtained using specific similarity measures and, thus, may also contain false positive subgraphs.

This work belongs to the second class of the algorithms. Most of these algorithms use *backtracking* to move through the built search tree and find appropriate combination of corresponding vertices of the source graph and the graph-pattern. Algorithms in this class include Ullmann algorithm [19], VF2 [20] (and also VF2 Plus [21] and VF3 [22]), TurboISO [23], CFL-Match [24], QuickSI [25], SPath [26] and others. These algorithms implement various techniques to decrease time needed for the matching process.

Exploiting Pruning Rules. The Ullmann algorithm uses *refining procedure* on each step of the algorithm by comparing degrees of corresponding neighbors of the added pair of vertices. VF2 [20] provides *feasibility rules* that are checked before a vertex is added to a graph-candidate. There rules check consistency of graph-candidates with this vertex and check for a sufficient number of vertices-neighbors of these graph-candidates. SPath [26] uses *neighborhood signature* for each vertex to store information about the surrounding vertices. These signatures are compared with the corresponding signatures of the query graph and are used for search space pruning before subgraph matching. TurboISO [23] compares quantity of *neighborhood labels* of corresponding vertices and prune out unpromising ones. CFL-Match [24] proposes a *compact-path-index* (CPI) structure presented as a tree which is built from the source graph vertices with the same labels as query graph vertices and then refined by exploiting matching operations.

Graph Pattern Modification. The Ullmann algorithm and VF2 [20] do not modify graph pattern and search its embeddings in the source graph. SPath [26] changes the way of graph query processing from vertex-at-a-time to *path-at-a-time*, which tends to be more cost-effective than traditional graph matching methods. TurboISO [23] presents a *NEC-tree* structure which merges similar vertices together and present a query graph as a tree. CFL-Match [24] transform a query into a set of *dense subgraphs*, *forests*, and *leaves*. The source graph in this algorithm is only probed for non-tree edge validation, whereas other query parts are checked in the CPI structure.

Optimizing Matching Order. The Ullmann algorithm [19] does not specify the matching order of the vertices, whereas VF2 [20] starts from a random query vertex and then recursively adds those vertices that are connected with the already matched ones. QuickSI [25] exploits an order which is based on the vertex label frequency, and the algorithm starts a process of matching from the least frequent ones. TurboISO [23] implements a concept of candidate region exploration and produces a matching order for every region where a NEC-tree was found. CFL-Match [24] present all candidates as a CPI-structure, where all the pattern embeddings are filtered and validated by traversing this tree structure.

The most of theoretical research of this problem was conducted specifically for ordinary graphs [18]; that is why the approaches of these algorithms have to be adapted to an *HP*-graph model. In particular, this paper presents an adaptation of a standard backtracking algorithm for subgraph matching, the Ullmann algorithm [19] and the VF2 algorithm [20], which are optimized for the multilayer structure of this graph model.

III. Graph-Matching Approach for *HP*-graphs

Let *Pol* be a set of all poles of the graph, including external poles and internal poles of vertices and hyperedges. Then, an *HP*-graph is an ordered triple $G = (P, V, W)$, where $P = \{\pi_1,\ldots,\pi_n\}$ is a set of external poles, $V = \{v_1,\ldots,v_m\}$ is a non-empty set of vertices, $W = \{w_1,\ldots,w_l\}$ is a set of hyperedges [7]. An example of the graph model is demonstrated on Fig. 1.

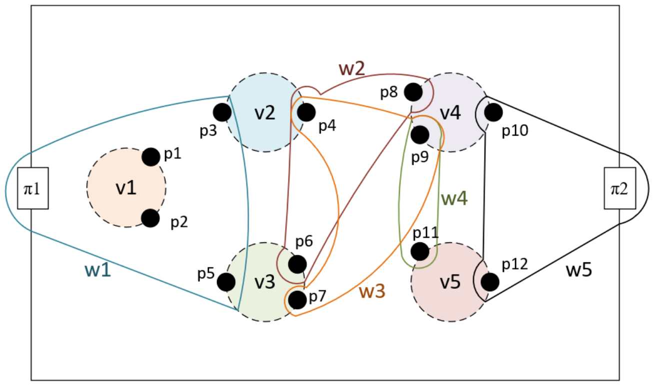

Fig. 1. Example of an *HP*-graph

In this figure external poles are represented by a set $P = \{\pi_1, \pi_2\}$, hyperedges by a set $W = \{w_1,\ldots,w_5\}$, and vertices by a set $V = \{v_1,\ldots,v_5\}$. A set *Pol* includes of the poles of the graph and is presented as $\{p_1,\ldots,p_{12}\}\cup\{\pi_1, \pi_2\}$.

Every hyperedge *w* of the *HP*-graph *G* can be presented by ordinary links, which are defined as a set $E_w = \{e_1,\ldots,e_n\}$, where every link ($e \in E_w$) is a pair of connected poles (p, r), where *p* is a source pole and *r* is a target pole of a link. An example of this decomposition is presented in Fig. 2. The hyperedge *w*2 defines a set $E_{w2} = \{(p4, p8), (p4, p6), (p6, p8)\}$.

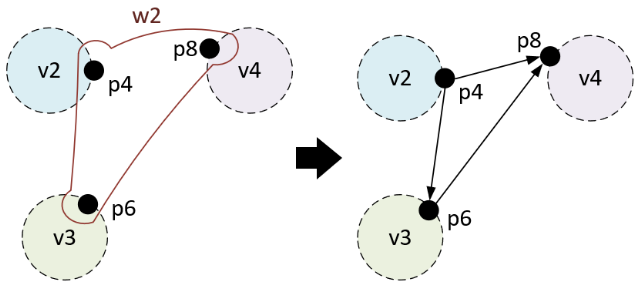

Fig. 2. Decomposition of the hyperedge w$_2$

Every vertex and hyperedge can also be decomposed by a new *HP*-graph, which is described in detail in [7].

A. *Definitions of a Subgraph and Isomorphism*

To determine subgraph matching operations, it is needed to give a definition to a subgraph of the *HP*-graph. An *HP*-graph $G' = (P', V', W')$ is a subgraph of an *HP*-graph $G = (P, V, W)$ iff *G'* is a part of the graph G ($P' \subset P$ & $(\forall v'\in V'\ \exists v \in V$: $[v' \subset v])$ & $W' \subset W$) and meets the condition (1) to make transformation operations possible [7]. A subgraph can contain vertices called *incomplete* whose sets of poles are only part of the sets of poles of the vertices of the original graph.

$$\forall w\in W(\exists v\in\ V'\backslash V'_{partial}\ ([Pol(w)\cap Pol(v)\neq\varnothing])\rightarrow w\in W'),\ (1)$$

the set $V'_{partial}$ is a set of the *incomplete vertices* in the graph, where $V'_{partial} \subset V'$.

To define the isomorphism mapping, it is necessary to establish one-to-one correspondences between the same type elements of graphs that preserve the incident relations. This, two *HP*-graphs $G = (P,V,W)$ and $G' = (P',V',W')$ are isomorphic iff there exists a bijection f: $2^{Pol(G)}\rightarrow 2^{Pol(G')}$ such that for $\forall t\in 2^{Pol(G)}$:

$$(t\in W\leftrightarrow f(t)\in W')\&(t\in V\leftrightarrow f(t)\in V')\&(t\in P\leftrightarrow f(t)\in P').$$

B. *A Multilayer Approach to Graph Matching*

As the graph model is proposed to store all the models together, search algorithms for this formalism have to be optimized for this task. A possible solution to this problem is to divide the *HP*-graph into two main levels: the level of vertices and hyperedges, and the level of poles and ordinary links between them. In this case, the search is started on the higher level, and when a candidate is found, it moves to the lower level, where a more detailed comparison of graph elements is performed.

Fig. 3(a) illustrates an example of a query graph *Q*, which is a pattern for subgraph matching for a data *G* from Fig. 1. As is seen, it contains 4 vertices, 2 hyperedges and 4 poles. Its higher (or first) level is presented in Fig. 3(b). It contains only 4 vertices and 2 hyperedges, whereas all the poles are eliminated. This layer is compared with the first layer of the graph *G* (Fig 4), and when a potential subgraph is found, the matrix of vertex correspondence is built.

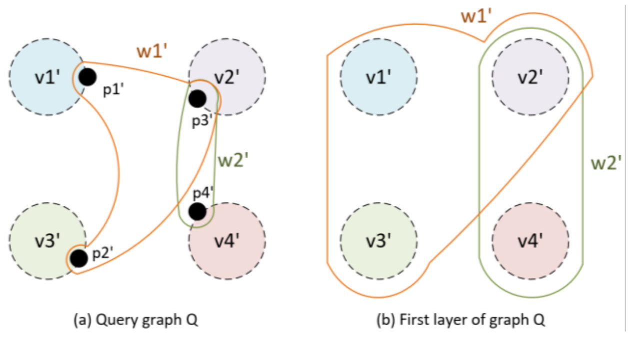

Fig. 3. Query graph *Q* and its first level

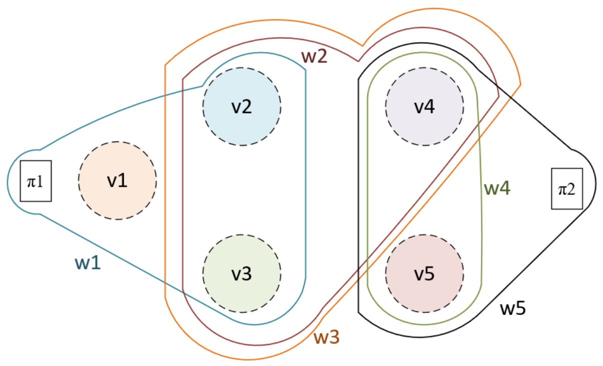

Fig. 4. First level of the graph *G*

The found correspondences between vertices of *Q* and *G* can be presented as a set {(*v*1', *v*2), (*v*3', *v*3), (*v*2', *v*4), (*v*4', *v*5)}. If a subgraph is found, the algorithm moves to the next level, where the corresponding hyperedges and their poles are compared.

All the candidate hyperedges are grouped by their incidence with each other depending on the poles which they consist of. For instance, hyperedges *w*1' and *w*2' are presented as a single group because of the pole *p*3' which both of them own. Thus, a corresponding pair (*w*3, *w*4) is also presented as a single group. All these groups are compared for exact isomorphism on the layer of poles and ordinary links. Fig. 5 demonstrates this layer for a pair of candidate groups (*w*1', *w*2') and (*w*3, *w*4). All these hyperedges are decomposed and only their poles and links are considered on this stage. As these graphs are identical, the found correspondences between poles of incident hyperedges of graph *Q* and *G*. can be presented as a set {(*p*3', *p*9), (*p*4', *p*11), (*p*2', *p*7), (*p*1', *p*4)}.

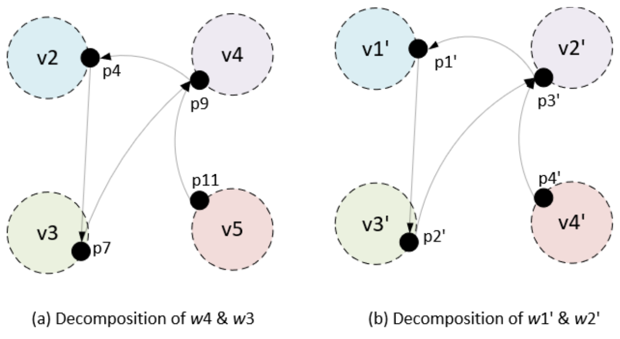

Fig. 5. Comparison of hyperedges (*w*3, *w*4) and (*w*1', *w*2')

If a validation on this hyperedge group is succeeded, the algorithm moves to the next group of hyperedges and validate them, until all the hyperedges are traversed. If a validation fails, the algorithm moves to the upper level and tries to find new pairs of vertices and hyperedges and validate them.

Lastly, the algorithm verifies that for every pole of the pattern graph only one pole of the source graph has been found. Otherwise, the found subgraph is considered as not isomorphic and the search continues.

IV. Backtracking Graph-Matching Algorithm based on the Multilayer Approach

The algorithm presented in this section uses as a basis a backtracking algorithm presented in [19]. This algorithm traverses a search tree using DFS until an isomorphic subgraph is found. If a pair of corresponding elements cannot be found at a certain step, a transition to an earlier step is carried out.

Considering the division of the subgraph matching into several levels, the search algorithm should be modified to perform the isomorphism search operation separately at the vertex level, separately at the hyperedge level, and separately at the level of poles and links.

Let *CompElems* define a set of compared elements: vertices, hyperedges or poles. Then, an algorithm for matching the corresponding sets of graph elements can be presented as follows:

Algorithm 1. Function *FindIsomorphism*(*G, Q, CompElems, args*)

```
  $M^0, M, H, F, k, d$ = InitializeValues(G, Q, CompElems, args);
  do:
    k = GetNextNonVisitedColumn(M, F, k);
    if (k == −1):
        if (d == 1):
          return null;
        else:
          MakeStepBack(F, d, M, k)
          continue;
    M = ChangeRowElementsToZerosExceptChosen(M, d, k);
    MakeStepForward(k, d, F, H, M)
  while ($d \leq |CompElems(Q)|$);
  return ValidateIsomorph(M’, CompElems(G), CompElems(Q))
```

This algorithm at the beginning initializes a matrix M^0 which defines possible candidates between corresponding elements of graphs. If $m^0_{ij} = 1$ then the *i*-th element of the first graph is a candidate for isomorphism for the *j*-th element of the second graph. Otherwise, they cannot form a pair of corresponding elements. At each step, the modification of this matrix is used to determine appropriate pairs of elements. Thus, it is needed to define rules for building this matrix for each set of *HP*-graph elements.

For vertices matching, external poles and vertices can be combined into one set and named as vertices (for simplification). Thus, the matrix $M^0 = |Q_V \cup Q_P| \times |G_V \cup G_P|$ is filled according to the rule (2). If this condition is not met, $m^0_{ij} = 0$.

$$m^0_{ij} = \{1 |\ Deg(v_{Gj}) \geq Deg(v_{Qi})\ \&\ Count(v_{Gj}) \geq Count(v_{Qi})\}, \quad (2)$$

Deg(*v*) is a number of hyperedges incident to the vertex *v*, *Count*(*v*) is a number of the vertex poles.

For hyperedges matching, the matrix $M^0 = |Q_W| \times |G_W|$ is filled according to the rule (3).

$$m^0_{ij} = \{1 |\ Vertices(w_{Gj}) \cong Vertices(w_{Qi})\}, \quad (3)$$

Vertices(*w*) is a set of vertices incident to the hyperedge *w*.

For poles matching, the matrix M^0 is created for each pair of grouped hyperedges; thus $M^0 = |Pol(W_{Ql})| \times |Pol(W_{Gm})|$. The matrix is filled according to the rule (4), considering that graphs *G* and *Q* on this stage only contain those hyperedges that are presented in the current groups.

$$\begin{aligned} m^0_{ij} = \{1 |\ & vertex(p_{Gj}) \cong vertex(p_{Qi})\ \&\ deg(p_{Gj}) \geq deg(p_{Qi})\ \& \\ & \&\ \forall edge(p_{Qi})\ \exists edge(p_{Gj})\ [edge(p_{Gj}) \cong edge(p_{Qi})]\ \& \quad (4) \\ & \&\ \forall edge(p_{Gj})\ \exists edge(p_{Qi})\ [edge(p_{Gj}) \cong edge(p_{Qi})]\ \}, \end{aligned}$$

vertex(*p*) is a vertex which contains a pole *p*, *edge*(*p*) is an edge which is incident to a pole *p*, *deg*(*p*) is a degree of a pole (a number of ordinary links incident to a pole).

Algorithm 2 illustrates how an isomorphic subgraph for the proposed graph structure can be found. Vectors *VCorr*, *WCorr* and *PolCorr* contain pairs of corresponding elements of the graphs. *FindIsomorphism* method is presented above and is assumed to have a possibility to continue the search from the position where the last candidate was found. For this purpose, the last argument for vertices and hyperedges isomorphism search is given to the algorithm (*VCorr* and *WCorr* respectively). *GroupByIncidence* combines the given hyperedges into groups, which represent incident edges.

Algorithm 2. Function *FindHPGraphIsomorphism*(*G, Q*)

```
VCorr = [$|V(Q) \cup P(Q)|$], WCorr = [$|W(Q)|$]; PolCorr = [$|Pol(Q)|$];
do:
  VCorr = FindIsomorphism(G, Q, $V(Q) \cup P(Q)$, VCorr);
  if (VCorr == $\varnothing$):
    continue;
  do:
    WCorr = FindIsomorphism(G, Q, W(Q), VCorr, WCorr);
    if (WCorr == $\varnothing$ & $|W(Q)| > 0$):
      break;
    incidentHyperedges = GroupByIncidence(WCorr, G, Q);
    for $\forall (W'_Q, W'_G) \in$ incidentHyperedges:
      polWCorr = FindIsomorphism(G, Q, Pol($W'_Q$), VCorr, WCorr);
      if (!PolCorr.TryAppend(polWCorr)):
        PolCorr = $\varnothing$;
        break;
    if (PolCorr != $\varnothing$ or $|W(Q)| == 0$):
      unlinkedCorr = MatchUnlinked(G, Q, PolCorr, VCorr, WCorr);
      GenerateAnswer(PolCorr, unlinkedCorr, VCorr, WCorr);
  while (PolCorr == $\varnothing$);
while (VCorr != $\varnothing$ & PolCorr == $\varnothing$)
```

The main idea of this algorithm is to incrementally shorten the search field. While the search for vertices traverses all the vertices of the original graph, the search for hyperedges only moves through those edges that are connected with the already chosen vertices and utilizes information about their correspondence with the vertices of the query graph. Pole matching is performed for each group of incident hyperedges, where a sufficient quantity of combinations is pruned out by exploiting information about the corresponding vertices and hyperedges. The algorithm also checks and matches the unlinked poles if they exist, which can be done in linear or close to linear time as all the corresponding vertices are already found. For simplicity, the algorithm is given for searching for the first isomorphic subgraph but can be transformed to searching for all embeddings of a pattern.

V. Exploiting Pruning Techniques of the Existing Algorithms

To optimize algorithms certain existing techniques can be used. Adaptation of the main techniques of the existing algorithms to the proposed graph model can prove the possibility of adapting these algorithms as a whole and improve the efficiency of the algorithm presented above.

A. Ullmann algorithm

Ullmann algorithm [19] is one of the first algorithms for subgraph matching. This algorithm uses a backtracking algorithm presented above and at each step it performs a *refinement procedure* to prune out unpromising pairs.

This algorithm is performed at each node of the search tree. It traverses the matrix M and converts a certain part of values from ones to zeros. The condition for preserving 1 is that if a vertex j of the original graph is a candidate of a vertex i of the pattern graph, then each neighbor of the vertex i must have at least one candidate among the neighbors of the vertex j. Otherwise, j cannot be a candidate for a vertex i.

This algorithm can be implemented for both vertex matching and pole matching to eliminate unpromising element pairs. The refining algorithm for vertices can be presented as follows:

Algorithm 3. Function *RefineV*(*G, Q, M*)

```
do:
   anyChanges = false;
   for $\forall i \in Range(|V(Q)|)$:
      if $(\neg\exists j: [M_{ij} = 1])$:
         return false;
      for $\forall j \in Range(|V(G)|)$:
         for $\forall x \in V(Q)\backslash\{v_{Qi}\}$ where $\exists w \in W(Q)\ [w \cap v_{Qi} \neq \varnothing \& w \cap x \neq \varnothing]$:
              if $(\neg\exists y \in V(G)\backslash\{v_{Gj}\}$
                 where $\exists w \in W(G)\ [w \cap v_{Gj} \neq \varnothing \& w \cap y \neq \varnothing] \& M_{xy} = 1)$:
                 $M_{ij} = 0$;
     anyChanges = true;
while (anyChanges);
return true;
```

The algorithm goes through all the neighbors of the current query vertex, which have at least one common hyperedge with this vertex, and checks whether a source graph contains a corresponding neighbor-vertex. The algorithm for poles looks similarly but poles and ordinary links are used instead of vertices and hyperedges.

B. VF2 algorithm

VF2 [20] has been proposed for performing subgraph matching on large graphs. Effective representation of data structures and the usage of feasibility rules significantly reduces both the average time complexity of the search and the amount of memory used.

The idea of the algorithm is to use special rules, called *feasibility rules*, at each node of the search tree to evaluate the feasibility of further progress on this branch of the tree before adding a pair of vertices to graph-candidates. There rules check consistency of graph-candidates and sufficiency of vertices-neighbors quantity of the graph-candidate. If all the checks are passed, the algorithm can move to the next level of the tree.

An approach of checking the feasibility rules can be applied on both vertex and pole layers. As a pole layer is presented as an ordinary graph, the feasibility rules from [20] can be used without any significant modifications. However, feasibility rules for a vertex layer have to be defined.

The first rule checks the consistency of the existent candidate graphs by checking correctness of connections with the already added vertices. Let $core_G$ be a list of found pair vertices for the graph G and $core_Q$ be a list of found pair vertices for the graph Q. Accordingly, let $conn_G$ be a list of vertices which already have a pair or have a connection to the current graph-candidate G' and $conn_Q$ be a similar list for the graph-candidate Q'. Then, the first rule can be presented as follows:

$$\begin{gathered}\forall n'[core_G[n'] \neq \varnothing \ \& \ n' \in Conn(G', n)]: \\ \exists m'[m' \in Conn(Q', m) \ \& \ core_Q[m'] = n'] \ \& \\ \& \ \forall m'[core_Q[m'] \neq \varnothing \ \& \ m' \in Conn(Q', m)]: \\ \exists n'[n' \in Conn(G', n) \ \& \ core_G[n'] = m'].\end{gathered}$$

$Conn(G, v)$ is a set of vertices of the candidate-graph G, which are connected to the vertex v.

Let PC define a set of vertices that can be connected to the vertex u, but the graph G does not include them; then it can be represented as follows:

$$PC(G, u) = \{v \mid v \in Conn(G, u) \ \& \ core_G[v] = \varnothing \ \& \ conn_G[v] \neq \varnothing\}.$$

Thus, a new rule, which compares numbers of newly added connections to graphs, appears:

$$|PC(G', n)| \geq |PC(Q', m)|.$$

The last rule performs a two-look-ahead in the searching process. Let N be a set of vertices which are connected to the target vertex but are not connected to the graph-candidate:

$$N(G, u) = \{v \mid v \in Conn(G, u) \ \& \ conn_G[v] = \varnothing\}.$$

Then, the last rule is presented by the condition:

$$|N(G', u)| > |N(Q', u)|.$$

The algorithm for traversing vertices can be presented as follows:

Algorithm 4. Procedure *RecurseV(G, Q, vectors)*

```
if $(\forall items \in vectors.core_Q[item \neq \varnothing])$:
   $poles_Q$ = RecurseW(vectors.$core_G$, vectors.$core_Q$, $\varnothing$, G, Q)
   if ($poles_Q$ != $\varnothing$):
      GenerateAnswer($poles_Q$);
   else:
      vectors = RestoreVectors(vectors);
else:
   P = GetAllCandidatePairs(vectors);
   for $\forall p \in P$:
      if (CheckVFisibilityRules(p, vectors, G, Q)):
         vectors = UpdateVectors(vectors, G, Q);
         RecurseV(G, Q, vectors);
   vectors = RestoreVectors(vectors);
```

C. Graph Pattern Modification Algorithms

The usage of algorithms such as TurboISO [23], CFL-Match [24] and other ones, that change a graph pattern, is complicated in the presented multilayer approach because these algorithms are made specifically for ordinary graphs. Their usage on the layer of vertices and hyperedges is a subject for the future research as it requires reformulation of their main aspects and ideas. Nevertheless, all these algorithms can be successfully used on the layer of poles and links and can find an isomorphic subgraph in the single-layer approach.

VI. Complexity of the Algorithms

The presented algorithms can decrease the complexity of subgraph search by implementing matching on different graph layers. The search field shortens at each stage whereas the usage of pruning rules can also eliminate unpromising combinations of elements. Table I shows computational complexity of the backtracking algorithm at its main stages.

TABLE I. Complexity of the Backtracking Algorithms

Algorithm	Best Case	Worst Case
Isomorphic Vertices Matching	O(N^2)	O(N×N!)
Isomorphic Hyperedges Matching	O(N^2)	O(N×N!)
Isomorphic Poles Matching	O(N^2)	O(N×N!)

The evaluation of the backtracking algorithms based on the Ullmann refinement is presented in Table II. As the algorithm of hyperedge matching does not implement this technique, its complexity stays the same.

TABLE II. Complexity of the Ullmann Algorithm

Algorithm	Best Case	Worst Case
Isomorphic Vertices Matching	O(N^3)	O(N^3×N!)
Isomorphic Hyperedges Matching	O(N^2)	O(N×N!)
Isomorphic Poles Matching	O(N^3)	O(N^3×N!)

The evaluation of the algorithms based on the VF2 approach is demonstrated in Table III. The modification of the *GetAllCandidatePairs* procedure slightly increases the worst-case complexity from N×N! to N^2×N! and the best-case complexity from N^2 to N^3.

TABLE III. Complexity of the VF2 Algorithm

Algorithm	Best Case	Worst Case
Isomorphic Vertices Matching	O(N^3)	O(N^2×N!)
Isomorphic Hyperedges Matching	O(N^3)	O(N^2×N!)
Isomorphic Poles Matching	O(N^3)	O(N^2×N!)

VII. Conclusion

This paper proposed a solution to the problem of identifying isomorphic subgraphs in *HP*-graphs. The proposed approach is based on implementing matching on different graph layers of the graph model and incrementally shortening the search field at each layer.

The designed algorithms for subgraph matching based on the multilayer approach and evaluations of their complexity are presented above. The proposed approach incrementally decreases the search field of the algorithm and helps to decrease its overall complexity. The usage of pruning rules of the existing algorithms can eliminate unpromising candidates at each stage of the proposed algorithm and thus, significantly shorten the size of the search tree.

It is planned to evaluate actual time complexity of these algorithms on various data sets and develop a visual modeling system using the proposed approach to subgraph matching.

References

[1] Koznov D.V. Metodologija i instrumentarij predmetno-orientirovannogo modelirovanija [Methodology and tools for domain-specific modeling]. Doctor Degree thesis, Saint-Petersburg, 2016, 430 p. (in Russian).

[2] Struchkov I.V. Formalizm dlja opisanija programmnyh sistem i vychislitel'nyh processov ciklicheskoj parallel'noj obrabotki dannyh real'nogo vremeni [A formalism for describing software systems and computational processes for cyclic parallel processing of real time data]. *Informacionno-upravljajushhie sistemy* [*Information and control systems*], 2006, no. 2, pp. 8-13.

[3] Courcelle B. Recognizable Sets of Graphs, Hypergraphs and Relational Structures: A Survey. In: *Developments in Language Theory. Lecture Notes in Computer Science*, 2005, vol. 3340, pp. 1-11.

[4] Power J., Tourlas K. Abstraction in Reasoning about Higraph-Based Systems. In: *Foundations of Software Science and Computation Structures. Lecture Notes in Computer Science*, 2003, vol. 2620, pp. 392-408.

[5] Sukhov A.O. Razrabotka instrumental'nyh sredstv sozdanija vizual'nyh predmetno-orientirovannyh jazykov [Development of tools for creating visual subject-oriented languages]. PhD thesis, Moscow, 2013, 256 p. (in Russian).

[6] Mikov A.I. Performance evaluation: textbook, 2013, 89 p.

[7] Suvorov N.M., Lyadova L.N. HP-Graph as a Basis of a DSM Platform Visual Model Editor. In: *Proceedings of the Institute for System Programming of the RAS*, 2020, vol. 32, no. 2, pp. 149-160.

[8] Parra F. Dean T. Survey of Graph Rewriting applied to Model Transformations. In: *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development*, 2014, pp. 431-441.

[9] Ehrig H., Ehrig K., Prange U., Taentzer G. Fundamentals of Algebraic Graph Transformation, 2006, 388 p.

[10] Yan X., Yu P.S., Han J. Graph Indexing: A Frequent Structurebased Approach. In: *Proceedings of SIGMOD*, 2004, pp. 335-346.

[11] Fan W. Graph pattern matching revised for social network analysis. In: *Proceedings ICDT*, 2012, pp. 8-21.

[12] Liu C., Lio B., Kropatsch W. Advances in Graph-based Pattern Recognition. *Pattern Recognition Letters*, 2017, vol. 87, 230 p.

[13] Pržulj N., Corneil D.G., Jurisica I. Efficient Estimation of Graphlet Frequency Distributions in Protein–protein Interaction Networks. *Bioinformatics*, 2006, vol. 22, no. 8, pp. 974-980.

[14] Han M., Kim H., Gu G., Park K., Han W. Efficient Subgraph Matching: Harmonizing Dynamic Programming, Adaptive Matching Order, and Failing Set Together. In: *Proceedings of SIGMOD*, 2019, pp. 1429-1446.

[15] Carletti V., Foggia P., Saggese A., Vento M. Challenging the Time Complexity of Exact Subgraph Isomorphism for Huge and Dense Graphs with VF3. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018, vol. 40, no. 4, pp. 804-818.

[16] Ren X., Wang J. Exploiting Vertex Relationships in Speending up Subgraph Isomorphism over Large Graphs. In: *Proceedings of the VLDB Endowment*, 2015, vol. 8, no. 5, pp. 617-628.

[17] Lee J., Han W., Kasperovics R., Lee J. An In-depth Comparison of Subgraph Isomorphism Algorithms in Graph Databases. In: *Proceedings if the VLDB Endowment*, 2012, vol. 6, no. 2, pp. 133-144.

[18] Seryi A.P., Lyadova L.N. An Approach to Graph Matching in the Component of Model Transformations. In: *Proceedings of the SYRCoSE 2013*. Kazan : 2013, pp. 41-46.

[19] Ullmann J.R. An Algorithm for Subgraph Isomorphism. In: *Journal of the ACM*, 1976, vol. 23, no. 1, pp. 31-42.

[20] Cordella L.P., Foggia P., Sansone C., Vento M. Performance evaluation of the VF graph matching algorithm. In: *Proceedings of 10th International Conference on Image Analysis and Processing*, 1999, pp. 1172-1177.

[21] Carletti V., Foggia P., Vento M. VF2 Plus: An Improved version of VF2 for Biological Graphs. In: *Graph-Based Representations in Pattern Recognition 2015. Lecture Notes in Computer Science*, 2015, vol. 9069, pp. 168-177.

[22] Carletti V., Foggia P., Saggese A., Vento M. Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with VF3. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018, vol. 40, no. 4, pp. 804-818.

[23] Han W., TurboISO: Towards UltraFast and Robust Subgraph Isomorphism Search in Large Graph Databases. In: *Proceedings of SIGMOD*, 2013, pp. 337-348.

[24] Bi F., Chang L., Lin X., Qin L., Zhang W. Efficient Subgraph Matching by Postponing Cartesian Products. In: *Proceedings of SIGMOD*, 2016, pp. 1199-1214.

[25] Shang H., Zhang Y., Lin X., Yu J.X. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. In: *Proceedings if the VLDB Endowment*, 2008, vol. 1, no. 1, pp. 364-375.

[26] Zhao P., Han J. On graph query optimization in large networks. In: *Proceedings if the VLDB Endowment*, 2010, vol. 3, pp. 340-351