Using metagraph approach for complex domains description

© Valeriy M. Chernenkiy, © Yuriy E. Gapanyuk, © Georgiy I. Revunkov,

© Yuriy T. Kaganov, © Yuriy S. Fedorenko, © Svetlana V. Minakova

Bauman Moscow State Technical University,
Moscow, Russia

chernen@bmstu.ru, gapyu@bmstu.ru, revunkov@bmstu.ru,

kaganov.y.t@bmstu.ru, fedyura11235@mail.ru, morgana_93@mail.ru

Abstract. This paper proposes an approach for complex domains description using complex network models with emergence. The advantages of metagraph approach are discussed. The formal definitions of the metagraph data model and metagraph agent model is given. The examples of data metagraph and metagraph rule agent are discussed. The metagraph and hypergraph models comparison is given. It is shown that the hypergraph model does not fully implement the emergence principle. The metagraph and hypernetwork models comparison is given. It is shown that the metagraph model is more flexible than hypernetwork model. Two examples of complex domains description using metagraph approach are discussed: neural network representation and modeling the polypeptide chain synthesis. The textual representation of metagraph model using predicate approach is given.

Keywords: metagraph, metavertex, metaedge, hypergraph, hypernetwork, neural network, polypeptide chain, lambda architecture.

1 Introduction

Currently, models based on complex networks are increasingly used in various fields of science from mathematics and computer science to biology and sociology. This is not surprising because the domains are becoming more and more complex.

Therefore, now it is important to offer not only a model that is capable of storing and processing Big Data but also a model that is capable of handling the complexity of data. That is why the development of a universal model for complex domains description is an actual task.

One of the varieties of such models is "complex networks with emergence". The emergent element means a whole that cannot be separated into its component parts.

As far as the authors know, currently there are two "complex networks with emergence" models: hypernetworks and metagraphs. The hypernetwork model is mature and it helps to understand many aspects of complex networks with an emergence.

But from the author's point of view, the metagraph model is more flexible and convenient for use in information systems.

This paper discusses the metagraph model and compares it with other complex graph models.

2 Complex networks models comparison

In this section, the metagraph model will be formally described and it will be compared with hypergraph and hypernetwork models.

2.1 Metagraph model formalization

A metagraph is a kind of complex network model, proposed by A. Basu and R. Blanning [1] and then adapted for information systems description by the authors [2]. According to [2]: $MG = \langle V, MV, E, ME \rangle$, where MG – metagraph; V – set of metagraph vertices; MV – set of metagraph metavertices; E – set of metagraph edges, ME – set of metagraph metaedges.

A metagraph vertex is described by the set of attributes: $v_i = \{atr_k\}, v_i \in V$, where v_i – metagraph vertex; atr_k – attribute.

A metagraph edge is described by the set of attributes, the source and destination vertices and edge direction flag: $e_i = \langle v_S, v_E, eo, \{atr_k\} \rangle, e_i \in E, eo = true|false$, where e_i – metagraph edge; v_S – source vertex (metavertex) of the edge; v_E – destination vertex (metavertex) of the edge; eo – edge direction flag (*eo=true* – directed edge, *eo=false* – undirected edge); atr_k – attribute.

The metagraph fragment: $MG_i = \{ev_j\}, ev_j \in (V \cup MV \cup E \cup ME)$, where MG_i – metagraph fragment; ev_j – an element that belongs to union of vertices, metavertices, edges and metaedges.

The metagraph metavertex: $mv_i = \langle \{atr_k\}, MG_j \rangle, mv_i \in MV$, where mv_i – metagraph metavertex belongs to set of metagraph metavertices *MV*; atr_k – attribute, MG_j – metagraph fragment.

Thus, a metavertex in addition to the attributes includes a fragment of the metagraph. The presence of private attributes and connections for a metavertex is distinguishing feature of a metagraph. It makes the definition of metagraph to be holonic – a metavertex may include a number of lower-level elements and in turn, may be included in a number of higher level elements.

From the general system theory point of view, a metavertex is a special case of the manifestation of the emergence principle, which means that the metavertex with its private attributes and connections becomes a whole that cannot be separated into its component parts. The example of metagraph is shown in figure 1.

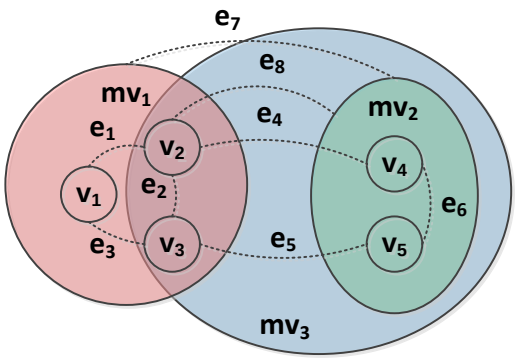


Figure 1 Example of metagraph

This example contains three metavertices: mv_1, mv_2, and mv_3. Metavertex mv_1 contains vertices v_1, v_2, v_3 and connecting them edges e_1, e_2, e_3. Metavertex mv_2 contains vertices v_4, v_5 and connecting them edge e_6. Edges e_4, e_5 are examples of edges connecting vertices v_2-v_4 and v_3-v_5 respectively and they are contained in different metavertices mv_1 and mv_2. Edge e_7 is an example of an edge connecting metavertices mv_1 and mv_2. Edge e_8 is an example of an edge connecting vertex v_2 and metavertex mv_2. Metavertex mv_3 contains metavertex mv_2, vertices v_2, v_3 and edge e_2 from metavertex mv_1 and also edges e_4, e_5, e_8 showing the holonic nature of the metagraph structure. Figure 1 shows that metagraph model allows describing complex data structures and it is the metavertex that allows implementing emergence principle in data structures.

The vertices, edges, and metavertices are used for data description and the metaedges are used for process description.

The metagraph metaedge: $me_i = \langle v_S, v_E, eo, \{atr_k\}, MG_j \rangle, me_i \in ME, eo = true|false$, where me_i – metagraph metaedge belongs to set of metagraph metaedges *ME*; v_S – source vertex (metavertex) of the metaedge; v_E – destination vertex (metavertex) of the metaedge; *eo* – metaedge direction flag (*eo=true* – directed metaedge, *eo=false* – undirected metaedge); atr_k – attribute, MG_j – metagraph fragment. The example of directed metaedge is shown in figure 2.

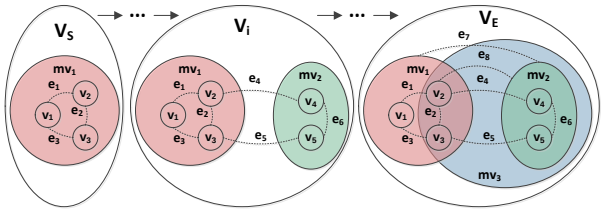


Figure 2 Example of directed metaedge

The directed metaedge contains metavertices v_S, … v_i, … v_E and connecting them edges. The source vertex contains a nested metagraph fragment. During the transition to the destination vertex, the nested metagraph fragment becomes more complex, as new vertices, edges, and metavertices are added. Thus, metaedge allows binding the stages of nested metagraph fragment development to the steps of the process described with metaedge.

2.2 Metagraph and hypergraph models comparison

In this section, the hypergraph model will be examined and compared with metagraph model. According to [3]: $HG = \langle V, HE \rangle, v_i \in V, he_j \in HE$, where HG – hypergraph; V – set of hypergraph vertices; HE – set of non-empty subsets of V called hyperedges; v_i – hypergraph vertex; he_j – hypergraph hyperedge.

A hypergraph may be directed or undirected. A hyperedge in an undirected hypergraph only includes vertices whereas, in a directed hypergraph, a hyperedge defines the order of traversal of vertices. The example of an undirected hypergraph is shown in figure 3.

This example contains three hyperedges: he_1, he_2, and he_3. Hyperedge he_1 contains vertices v_1, v_2, v_4, v_5. Hyperedge he_2 contains vertices v_2 and v_3. Hyperedge he_3 contains vertices v_4 and v_5. Hyperedges he_1 and he_2 have a common vertex v_2. All vertices of hyperedge he_3 are also vertices of hyperedge he_1.

Comparing metagraph and hypergraph models it should be noted that the metagraph model is more expressive then the hypergraph model. According to figures 1 and 3 it is possible to note some similarities between the metagraph metavertex and the hypergraph hyperedge, but the metagraph offers more details and clarity because the metavertex explicitly defines metavertices, vertices and edges inclusion, whereas the hyperedge does not. The inclusion of hyperedge he_3 in hyperedge he_1 in fig. 3 is only graphical and informal, because according to hypergraph definition a hyperedge inclusion operation is not explicitly defined.

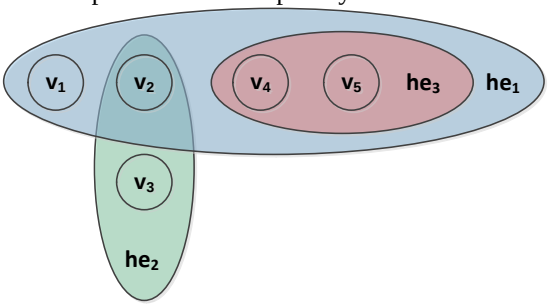


Figure 3 Example of undirected hypergraph

Thus the metagraph is a holonic graph model whereas the hypergraph is a near flat graph model that does not fully implement the emergence principle. Therefore, hypergraph model doesn't fit well for complex data structures description.

2.3 Metagraph and hypernetwork models comparison

The amazing fact is that the hypernetwork model was invented twice. The first time the hypernetwork model was invented by Professor Vladimir Popkov with colleagues in 1980s. Professor V. Popkov proposes several kinds of hypernetwork models with complex

formalization and therefore only main ideas of hypernetworks will be discussed in this section. According to [4] given the hypergraphs $PS \equiv WS_0, WS_1, WS_2, \cdots WS_K$. The hypergraph $PS \equiv WS_0$ is called primary network. The hypergraph WS_i is called a secondary network of order i. Also given the sequence of mappings between networks of different orders: $WS_K \overset{\Phi_K}{-} WS_{K-1} \overset{\Phi_{K-1}}{-} \cdots WS_1 \overset{\Phi_1}{\rightarrow} PS$. Then the hierarchical abstract hypernetwork of order K may be defined as $AS^K = \langle PS, WS_1, \cdots WS_K; \Phi_1, \cdots \Phi_K \rangle$. The emergence in this model occurs because of the mappings Φ_i between the layers of hypergraphs.

The second time the hypernetwork model was proposed by Professor Jeffrey Johnson in his monograph [5] in 2013. The main idea of Professor J. Johnson variant of hypernetwork model is the idea of hypersimplex (the term is adopted from polyhedral combinatorics). According to [5], a hypersimplex is an ordered set of vertices with an explicit n-ary relation and hypernetwork is a set of hypersimplices. In the hierarchical system, the hypersimplex combines k elements at the N level (base) with one element at the N+1 level (apex). Thus, hypersimplex establishes an emergence between two adjoining levels.

The example of hypernetwork that combines the ideas of two approaches is shown in figure 4.

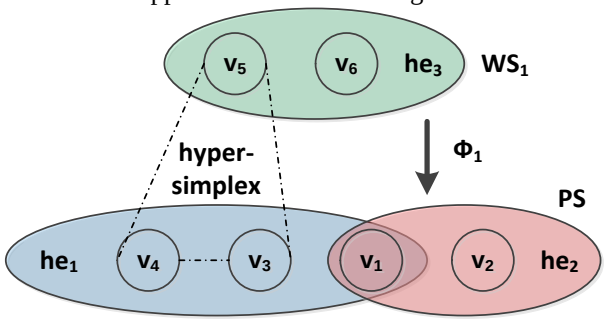


Figure 4 Example of hypernetwork

The primary network PS is formed by the vertices of hyperedges he_1 and he_2. The first level WS_1 of secondary network is formed by the vertices of hyperedge he_3. Mapping Φ_1 is shown with an arrow. The hypersimplex is emphasized with the dash-dotted line. The hypersimplex is formed by the base (vertices v_3 and v_4 of PS) and apex (vertex v_5 of WS_1).

The hypernetwork model became popular for complex domains description. For example, Professor Konstantin Anokhin [6] proposes a new fundamental theory of the organization of higher brain functions. According to this theory, biological neural networks (connectomes) are organized into cognitive hypernetworks (cognitomes). Vertices of cognitome form COGs (Gognitive Groups). Each COG may be represented as hypersimplex. The base of COG is a set of the vertices of underlying neural networks, and its apex is a vertex possessing a new quality at the macrolevel of cognitive hypernetworks. Thus, apex combines the base elements and emergence appears.

It should be noted that unlike the relatively simple hypergraph model the hypernetwork model is a full model with emergence. Consider the differences between the hypernetwork and metagraph models.

According to the definition of a hypernetwork it is a a layered description of graphs. It is assumed that the hypergraphs may be divided into homogeneous layers and then mapped with mappings or combined with hypersimplices. Metagraph approach is more flexible. It allows combining arbitrary elements that may be layered or not using metavertices.

Comparing the hypernetwork and metagraph models we can make the following notes:

- Hypernetwork model may be considered as "horizontal" or layer-oriented. The emergence appears between adjoining levels using hypersimplices. The metagraph model may be considered as "vertical" or aspect-oriented. The emergence appears between any levels using metavertices.
- In hypernetwork model, the elements are organized using hypergraphs inside layers and using mappings or hypersimplices between layers. In metagraph model, metavertices are used for organizing elements both inside layers and between layers. Hypersimplex may be considered as a special case of metavertex.
- Metagraph model allows organizing the results of previous organizations. The fragments of the flat graph may be organized into metavertices, metavertices may be organized in higher-level metavertices and so on. The metavertex organization is more flexible than hypersimplex organization because hypersimplex assumes base and apex usage and metavertex may include general form graph.
- Metavertex may represent a separate aspect of the organization. The same fragments of the flat graph may be included in different metavertices whether these metavertices are used for modeling different aspects.

Thus, we can draw a conclusion that metagraph model is more flexible than hypernetwork model.

However, it must be emphasized that the hypernetwork and metagraph models are only different formal descriptions of the same processes that occur in the networking with the emergence.

From the historical point of view, the hypernetwork model was the first complex network with an emergence model and it helps to understand many aspects of complex networks with an emergence.

3 Metagraph model processing

The metagraph model is designed for complex data and process description. But it is not intended for data transformation. To solve this issue, the metagraph agent (ag^{MG}) designed for data transformation is proposed. There are two kinds of metagraph agents: the metagraph function agent (ag^F) and the metagraph rule agent (ag^R). Thus $ag^{MG} = ag^F | ag^R$.

The metagraph function agent serves as a function with input and output parameter in form of metagraph:

$$ag^F = \langle MG_{IN}, MG_{OUT}, AST \rangle, \quad (1)$$

where ag^F – metagraph function agent; MG_{IN} – input parameter metagraph; MG_{OUT} – output parameter metagraph; AST – abstract syntax tree of metagraph function agent in form of metagraph.

The metagraph rule agent is rule-based: $\mathrm{ag}^{\mathrm{R}} = \langle \mathrm{MG}, \mathrm{R}, \mathrm{AG}^{\mathrm{ST}} \rangle, \mathrm{R} = \{\mathrm{r_i}\}, \mathrm{r_i}: \mathrm{MG_j} \to \mathrm{OP}^{\mathrm{MG}}$, where ag^R – metagraph rule agent; MG – working metagraph, a metagraph on the basis of which the rules of agent are performed; R – set of rules r_i; AG^{ST} – start condition (metagraph fragment for start rule check or start rule); MG_j – a metagraph fragment on the basis of which the rule is performed; OP^{MG} – set of actions performed on metagraph.

The antecedent of the rule is a condition over metagraph fragment, the consequent of the rule is a set of actions performed on metagraph. Rules can be divided into open and closed.

The consequent of the open rule is not permitted to change metagraph fragment occurring in rule antecedent. In this case, the input and output metagraph fragments may be separated. The open rule is similar to the template that generates the output metagraph based on the input metagraph.

The consequent of the closed rule is permitted to change metagraph fragment occurring in rule antecedent. The metagraph fragment changing in rule consequent cause to trigger the antecedents of other rules bound to the same metagraph fragment. But incorrectly designed closed rules system can lead to an infinite loop of metagraph rule agent.

If the agent contains only open rules it is called an open agent. If the agent contains only closed rules it is called a closed agent.

Thus, metagraph rule agent can generate the output metagraph based on the input metagraph (using open rules) or can modify the single metagraph (using closed rules). The example of metagraph rule agent is shown in figure 5.

The metagraph rule agent "metagraph rule agent 1" is represented as a metagraph metavertex. According to the definition it is bound to the working metagraph MG_1 – a metagraph on the basis of which the rules of the agent are performed. This binding is shown with edge e_4.

The metagraph rule agent description contains inner metavertices corresponds to agent rules (rule 1 … rule N). Each rule metavertex contains antecedent and consequent inner vertices. In given example mv_2 metavertex bound with antecedent which is shown with edge e_2 and mv_3 metavertex bound with consequent which is shown with edge e_3. Antecedent conditions and consequent actions are defined in form of attributes bound to antecedent and consequent corresponding vertices.

The start condition is given in form of attribute "start=true". If the start condition is defined as a start metagraph fragment then the edge bound start metagraph fragment to agent metavertex (edge e_1 in given example) is annotated with attribute "start=true". If the start condition is defined as a start rule than the rule metavertex is annotated with attribute "start=true" (rule 1 in given example). Figure 5 shows both cases corresponding to the start metagraph fragment and to the start rule.

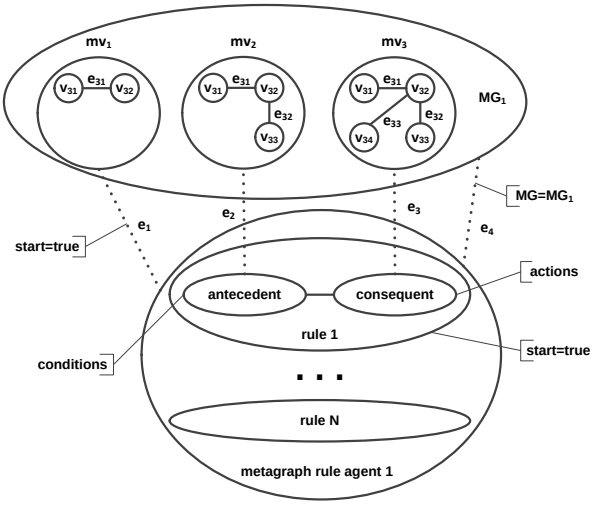


Figure 5 Example of metagraph rule agent

The distinguishing feature of metagraph agent is its homoiconicity which means that it can be a data structure for itself. This is due to the fact that according to definition metagraph agent may be represented as a set of metagraph fragments and this set can be combined in a single metagraph. Thus, the metagraph agent can change the structure of other metagraph agents.

4 The examples of complex domains description using metagraph approach

In this section, we give two examples of complex domains description using metagraph approach.

4.1 Using metagraph approach for neural network representation

This subsection is based on our paper [7]. We begin with simple perceptron representation using metagraph model. According to the Rosenblatt perceptron model [8], a conventional perceptron consists of three elements: S, A and R.

The layer of sensors (S) is an array of input signals. The associative layer (A) is a collection of intermediate elements which are triggered if a particular set of input signals is activated at the same time. The adder (R) is started when a particular collection of A-elements is activated concurrently.

According to the notation adopted in the M. Minsky and S. Papert perceptron model [8], the value of a signal on an A-element can be represented as a boolean predicate $\varphi(S)$, and the value of a signal in the adder layer as a predicate $\psi(A, W)$. According to [8], a function that takes either 0 or 1 is regarded as a boolean predicate.

Depending on the particular type of perceptron, the form of predicates $\varphi(S)$ and $\psi(A, W)$ can be different. Usually, predicate $\varphi(S)$ is used to check whether the total input signal from sensors exceeds a certain threshold or not. Also predicate $\psi(A, W)$ (where W is a weight vector) is used to see if the weighted sum from A-elements exceeds a particular threshold.

In our case, the actual form of predicates is not important. What is important is that the structure of *φ(S)* and *ψ(A, W)* can be represented as an abstract syntactic tree. Then we can represent the perceptron structure as a combination of metagraph function agents. Each predicate can be represented as a kind of the formula 1: $\varphi^F = \langle \mathrm{S}, \mathrm{A}, AST^{\varphi} \rangle, \psi^F = \langle \langle \{\varphi^F\}, W \rangle, \mathrm{R}, AST^{\psi} \rangle$. This representation is shown in figure 6.

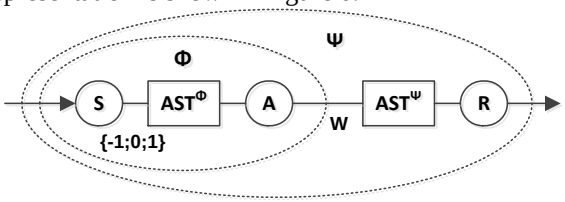


Figure 6 The perceptron representation as a combination of metagraph function agents

An A-element can be represented as a function agent φ^F. The input parameter is the value vector S, the output parameter is the value vector A. The description of the perceptron is similar to the description of the function agent ψ^F. The input parameter is a the metagraph representation of a tuple holding the description of A-elements as agent-functions φ^F and vector W. The output parameter is the amplitude of output signal R.

The description of functions can contain other parameters, e.g., threshold values, but we assume that these parameters are included in the description of the abstract syntactic tree.

Thus, we can describe the perceptron structure as a combination of metagraph function agents. Now we describe neural network operation using metagraph rule agents which are shown in figure 7.

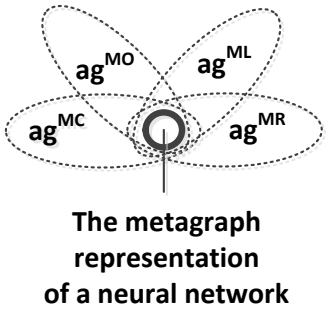


Figure 7 The structure of metagraph rule agents for neural network operation representation

The metagraph representation of neural network may be created similarly to the previously reviewed perceptron approach. Such a representation is a separate task that depends on neural network topology.

In order to provide a neural network operation the following agents are used:

- ag^{MC} – the agent responsible for the creation of the network;
- ag^{MO} – the agent responsible for the modification of the network;
- ag^{ML} – the agent responsible for the learning of the network;
- ag^{MR} – the agent responsible for the execution of the network.

In figure 7 the agents are shown as metavertices by dotted-line ovals.

The network-creating agent ag^{MC} implements the rules of creating an original neural network topology. The agent holds both the rules of creating separate neurons and rules of connecting neurons into a network. In particular, the agent generates abstract syntactic trees of metagraph function agents φ^F and ψ^F.

The network-modification agent ag^{MO} holds the rules of modification the network topology in process of operation. It is especially important for neural networks with variable topology such as HyperNEAT and SOINN.

The network-learning agent ag^{ML} implements a particular learning algorithm. As a result of learning the changed weights are written in the metagraph representation of the neural network. It is possible to implement a few learning algorithms by using different sets of rules for agent ag^{ML}.

The network-executing agent ag^{MR} is responsible for the start and operation of the trained neural network.

The agents can work separately or jointly which may be especially important in the case of variable topologies. For example when a HyperNEAT or SOINN network is trained, agent ag^{ML} can call the rules of agent ag^{MO} to change the network topology in the process of learning.

In fact, each agent uses its rules to implement a specific program "machine". The use of the metagraph approach allows us to implement the "multi-machine" principle: a few agents having different goals implement different operations on the same data structure.

Thus, we can draw a conclusion that metagraph approach helps to describe both the structure of separate neurons and the structure of neural network operation.

4.2 Using metagraph approach for modeling the polypeptide chain synthesis

Molecular biology is considered to be one of the most difficult to study topics of biological science. It's hard to believe that the complexity of functioning of the biological cell invisible to the human eye exceeds the complexity of functioning of a large ERP-system, which can contain thousands of business processes. The difficulty of studying biological processes is also due to the fact that in studying it is impossible to abstract from the physical and chemical features that accompany these processes. Therefore, the development of learning software that helps to better understand even one complex process is a valid task.

We will review the process of synthesis of a polypeptide chain which is also called "translation" in molecular biology. Translation is an essential part of the protein biosynthesis. This process is very valid from an educational point of view because protein biosynthesis is considered in almost all textbooks of molecular biology.

The translation process is very complicated and in this section, we review it in a simplified way.

The first main actor of the translation process is messenger RNA or mRNA, which may be represented as

a chain of codons. The second main actor of the translation process is ribosome consisting of the large subunit and the small subunit. The small subunit is responsible for reading information from mRNA and large subunit is responsible for generating fragments of the polypeptide chain.

According to [9] the translation process consists of three stages.

The first stage is initiation. At this stage, the ribosome assembles around the target mRNA. The small subunit is attached at the start codon.

The second stage is elongation. The small subunit reads information from the current codon. Using this information, the large subunit generates the fragment of the polypeptide chain. After that ribosome moves (translocates) to the next mRNA codon.

The third stage is termination. When the stop codon is reached, the ribosome releases the synthesized polypeptide chain. Under some conditions, the ribosome may be disassembled.

In this section, we use metagraph approach for translation process modeling. The representation is shown in figure 8.

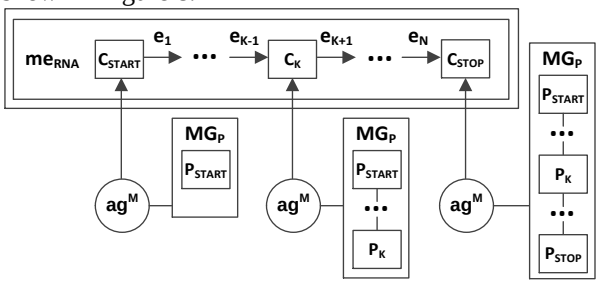


Figure 8 The representation of the polypeptide chain synthesis (translation) process based on metagraph approach

The mRNA is shown in figure 8 as metaedge $me_{RNA} = \langle C_{START}, C_{STOP}, eo = true, \{atr_k\}, MG_{RNA} \rangle$, where C_{START} – source metavertex (start codon); C_{STOP} – destination metavertex (stop codon); *eo=true* – directed metaedge; atr_k – attribute, MG_{RNA} – metagraph fragment, containing inner codons of mRNA (C_K) linked with edges.

The codon (shown in figure 8 as an elementary vertex) may also be represented as metavertex, containing inner vertices and edges according to the required level of detail.

Ribosome may be represented as metagraph rule agent $\text{ag}^{\text{RB}} = \langle \text{me}_{\text{RNA}}, \text{R}, C_{START} \rangle, \text{R} = \{\text{r}_\text{i}\}, \text{r}_\text{i}: \text{C}_\text{K} \rightarrow \text{P}_\text{K}$, where me_{RNA} – mRNA metaedge representation used as working metagraph; R – set of rules r_i; C_{START} – start codon used as start agent condition; C_K – codon on the basis of which the rule is performed; P_K – the added fragment of polypeptide chain.

The antecedent of the rule approximately corresponds to the small subunit of ribosome modeling. The consequent of the rule approximately corresponds to the large subunit of ribosome modeling.

Agent ag^{RB} is open agent generating output metagraph MG_P based on input metaedge me_{RNA}. The input and output metagraph fragments don't contain common elements.

While processing codons of mRNA agent ag^{RB} sequentially adds fragments of the polypeptide chain P_K to the output metagraph MG_P. Vertices P_K are connected with undirected edges.

The process represented in figure 8 is very high-level. But metagraph approach allows representing related processes with different levels of abstraction.

For example, for each codon or peptide, we can link metavertex with its inner representation. And we can modify this representation during translation process using metagraph agents.

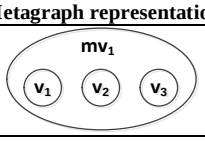
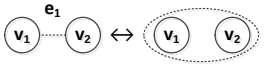
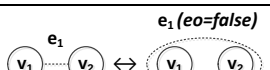
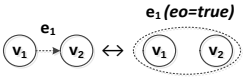
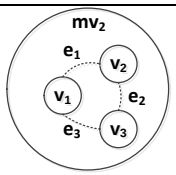
Thus, the metagraph approach allows us to represent a model of polypeptide chain synthesis which can be the basis for the learning software.

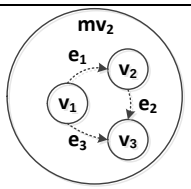
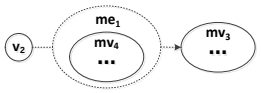
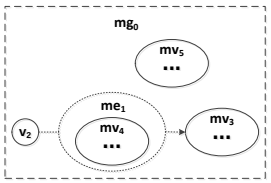
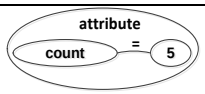
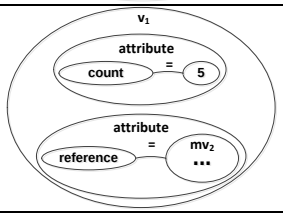
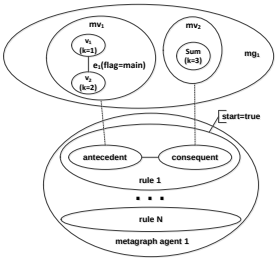
5 The textual representation of metagraph model

In previous sections, the formal definition and graphical examples of metagraph model were defined. But to successfully operate with metagraph model we also need textual representation. As such a representation, we use a logical predicate model that is close to logical programming languages e.g. Prolog. Logical predicates used in this section and boolean predicates used in subsection 4.1 should not be confused.

The classical Prolog uses following form of predicate: $predicate(atom_1, atom_2, \cdots, atom_N)$. We used an extended form of predicate where along with atoms predicate can also include key-value pairs and nested predicates: $predicate(atom, \cdots, key = value, \cdots, predicate(\cdots), \cdots)$. The mapping of metagraph model fragments into predicate representation is shown in Table 1.

Table 1 The textual representation of metagraph model

№	Metagraph representation	Textual representation
1		Metavertex(Name=mv_1, v_1, v_2, v_3)
2		Edge(Name=e_1, v_1, v_2)
3		Edge(Name=e_1, v_1, v_2, eo=false)
4		1. Edge(Name=e_1, v_1, v_2, eo=true) 2. Edge(Name=e_1, v_S=v_1, v_E=v_2, eo=true)
5		Metavertex(Name=mv_2, v_1, v_2, v_3, Edge (Name=e_1, v_1, v_2), Edge(Name=e_2, v_2, v_3), Edge(Name=e_3, v_1, v_3))

6		Metavertex(Name=mv_2, v_1, v_2, v_3, Edge(Name=e_1, v_S=v_1, v_E=v_2, eo=true), Edge(Name=e_2, v_S=v_2, v_E=v_3, eo=true), Edge(Name=e_3, v_S=v_1, v_E=v_3, eo=true))
7		Metaedge(Name=me_1, v_S=v_2, v_E=mv_3, Metavertex(Name=mv_4, …), eo=true)
8		Metagraph(Name=mg_0, Vertex(Name=v_2, …), Metavertex(Name=mv_3, …), Metavertex(Name=mv_5, …), Metaedge(Name=me_1, v_S=v_2, v_E=mv_3, Metavertex(Name=mv_4, …), eo=true))
9		Attribute(count, 5)
10		Vertex(Name=v_1, Attribute(count, 5), Attribute(reference, mv2))
11		Agent(Name= metagraph agent 1', WorkMetagraph=mg1, Rules(Rule(Name=rule 1', start=true, Condition (WorkMetagraph=mv_1, Vertex(Name=v_1, Attribute(k, $k1)), Vertex(Name=$v_2$, Attribute(k, $k2)), Edge($v_1$, v_2, Attribute(flag, main))) Action (WorkMetagraph=mv_2, Add(Vertex(Name=Sum, Attribute(k, Eval($k1+$k2)))))), Rule(…) …))

Case 1 shows the example of metavertex mv_1 which contains three nested disjoint vertices v_1, v_2, and v_3. The predicate corresponds to metavertex, nested vertices are isomorphic to atoms that are parameters of the predicate. As the name of the predicate, "Metavertex" is used as the corresponding element of metagraph model. Key-value parameter "Name" is used to set the name of metavertex. This case is the simplest, since nested vertices are disjoint, and metavertex in this case is isomorphic to the hypergraph hyperedge.

Case 2 shows metagraph edge which may be represented as a special case of metavertex containing source and destination vertices. This case is also isomorphic to the hypergraph hyperedge. The metagraph edge is represented as a predicate with the name "Edge". The source and destination vertices are represented as predicate atom parameters.

Case 3 also shows metagraph edge which fully complies with the formal definition of undirected edge including direction flag parameter.

Case 4 shows an example of directed edge. Direction flag parameter is also used. The source and destination vertices may be represented as predicate atom parameters (case 4.1) or as predicate key-value parameters (case 4.2).

Case 5 shows an example of metavertex mv_1 which contains three nested vertices v_1, v_2 and v_3 joined with undirected edges e_1, e_2, and e_3. Edges are represented with separate predicates that are nested to the metavertex predicate. Case 6 is similar to case 5 unless edges e_1, e_2, and e_3 are directed.

Case 7 shows an example of directed metaedge me_1 which joins vertex v_2 and metavertex mv_3 and contains metavertex mv_4. The metaedge is represented as a predicate with the name "Metaedge".

Case 8 shows an example of metagraph fragment mg_0 which contains vertex v_2, metavertices mv_3 and mv_5 and metaedge me_1 which joins vertex v_2 and metavertex mv_3 and contains metavertex mv4. The metagraph fragment is represented as a predicate with the name "Metagraph", the vertex as a predicate with the name "Vertex".

The attribute may be represented as a special case of metavertex containing name and value. Case 9 shows simple numeric attribute representation. Case 10 shows an example of vertex v_1 containing numeric attribute and reference attribute that refers to the metavertex mv_2. The attribute is represented as a predicate with the name "Attribute".

Case 11 shows an example of metagraph rule agent "metagraph agent 1" representation (the predicate with the name "Agent" is used). As a work metagraph mg_1 is used (parameter "WorkMetagraph"). The "Rules" predicate contains rules definition (nested predicate "Rule" is used). As a start rule "rule 1" is used which is defined by "start=true" parameter. Predicate "Condition" corresponds to the rule condition. Parameter "WorkMetagraph" contains a reference to the tested metavertex mv_1. The condition tests that metavertex mv_1 contains vertices v_1 and v_2 with attribute k. Founded values of k attribute of vertices v_1 and v_2 are assigned to the $k1 and $k2 variables. Vertices v_1 and v_2 should be joined with edge containing attribute "flag=main". If condition holds and metagraph fragment is found then actions are performed (actions are defined by predicate "Action"). Parameter "WorkMetagraph" contains a reference to the result metavertex mv_2. The example action contains adding new elements (that is defined by predicate "Add"). The vertex "Sum" is added containing attribute "k=$k1+$k2". Predicate "Eval" is used to define the calculated expression.

Thus, we defined a predicate description of all the main elements of metagraph data model.

The proposed predicate model is homoiconic. Since predicate approach is used both for metagraph data model definition and for metagraph agents definition then high-level metagraph agents may change the structure of low-level metagraph agents by modifying their predicate definition.

The textual representation of metagraph model may be used for storing metagraph model elements in relational or NoSQL databases.

It should be noted that metagraph model is well compatible with the Big Data approach. Nowadays the lambda architecture described in [10] is considered to be a classic approach.

The textual representation of metagraph model is the base for processing metagraph data on all layers of the lambda architecture. On the batch layer, the textual representation is used for storing in master dataset. On the serving layer, the textual representation helps to construct the batch views. On the speed layer, the textual representation helps to construct the real-time views. Batch and real-time views may be constructed using metagraph agents.

6 Conclusion

Nowadays complex network models have become popular for complex domains description.

The metagraph model is a kind of complex network model. The emergence in metagraph model is established using metavertices and metaedges.

The hypergraph model does not fully implement the emergence principle.

The hypernetwork model fully implements the emergence principle using hypersimplices. The metagraph model is more flexible than hypernetwork model.

For metagraph model processing, the metagraph function agents and the metagraph rule agents are used.

Two examples of complex domains description using metagraph approach are discussed: neural network representation and modeling the polypeptide chain synthesis. Metagraph approach helps to describe complex domains in a unified way.

The textual representation of metagraph model may be used for storing metagraph model elements in relational or NoSQL databases.

The metagraph model is well compatible with the Big Data approach, in particular with the lambda architecture.

References

[1] Basu A., Blanning R. Metagraphs and their applications. Springer, New York (2007)

[2] E. Samohvalov, G. Revunkov, Yu. Gapanyuk, “Metagraphs for describing semantics and pragmatics of information systems,” in Herald of Bauman Moscow State Technical University, vol. 1(100), pp.83-99, (2015)

[3] Vitaly I. Voloshin. Introduction to Graph and Hypergraph Theory. Nova Science Publishers, Inc., (2009)

[4] Akhmediyarova, A.T., Kuandykova, J.R., Kubekov, B.S., Utepbergenov, I.T., Popkov, V.K.: Objective of Modeling and Computation of City Electric Transportation Networks Properties. In: International Conference on Information Science and Management Engineering (Icisme 2015), pp. 106–111, Destech Publications, Phuket (2015)

[5] Johnson, J.: Hypernetworks in the Science of Complex Systems. Imperial College Press, London (2013)

[6] Anokhin, K.V.: Cognitom: theory of realized degrees of freedom in the brain. In: The Report Given at Fifth International Conference on Cognitive Science, Kaliningrad (2012)

[7] Fedorenko, Yu.S., Gapanyuk, Yu.E. Multilevel neural net adaptive models using the metagraph approach. Optical Memory and Neural Networks. Volume 25, Issue 4, pp. 228–235 (2016)

[8] Minsky, M.L., Papert, S.A. Perceptrons. The MIT Press (1988)

[9] Samish, I. Computational Protein Design. Springer Science+Business Media, New York (2017)

[10] Marz, N., Warren, J. Big Data. Principles and best practices of scalable realtime data systems. Manning, New York (2015)