Information Processing Letters 116 (2016) 611–617

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

A technique for the concept-based detection of functional modules in an interaction network

Jong-Min Park [a,1], Hyung-Jeong Yang [b,1], Jae-Dong Yang [a,*], Dong-Hoon Choi [c]

[a] Chonbuk National University, Chonju, 561-756, South Korea
[b] Chonnam National University, Gwangju, 500-757, South Korea
[c] Korea Institute of Science and Technology Information, Daejeon, 305-335, South Korea

ARTICLE INFO

Article history:
Received 12 March 2015
Received in revised form 31 October 2015
Accepted 22 November 2015
Available online 6 May 2016
Communicated by W.-L. Hsu

Keywords:
Information retrieval
Concept-based detection
Functional modules
Interaction network

ABSTRACT

In this paper, we propose a concept-based detection of functional modules in an interaction network. This method makes it possible to detect functional modules that are conceptually identical to users' needs.

1. Introduction

Functional modules such as complexes or chemical compounds are responsible for particular roles in an interaction network. The following is an example rule used to detect the pattern of the functional module, *Parkinson's disease*.

Parkinson's disease

$\rightarrow$ *Inhibition of transmitter release* |

Absence of Lewy bodies

Absence of Lewy bodies

$$\rightarrow \ldots^{*} < \text{UBCH7}, \text{PARK2}, \text{“inactivate”} >^{*}$$

$$< \text{PARK2}, \text{SNCAIP}, \text{“inactivate”} >$$

The meaning of the rule is that *Inhibition of transmitter release* or *Absence of Lewy bodies* may be regarded as *Parkinson's disease*. In the sequel, if UBCH7 inactivates PARK2 and PARK2, in turn, inactivates SNCAIP, this pattern may be detected as a functional sub-module, *Absence of Lewy body*. UBCH7, PARK2 and SNCAIP are protein names.

A large number of methods to search modules with similar functions or structures have been developed for complex networks [5–7]. These methods rely on the module search generally corresponding to dense sub-graphs in certain scales with modularity functions, modularity densities or multiresolution method. However, these methods cannot find various structured modules whose components play different roles but are conceptually connected to each other.

In this paper, we propose a new method to detect the functional modules based on the notion of concept modules. A concept module is basically a set of functional modules, which share not only syntactically the same structure but also conceptually the same meaning with each other. It is utilized to match such functional modules as its instances. Its pattern is defined by an expression rule composed of triples and operators between them. The rule may

* Corresponding author.
E-mail addresses: jmpark@chonbuk.ac.kr (J.-M. Park), hjyang@jnu.ac.kr (H.-J. Yang), jdyang@jbnu.ac.kr (J.-D. Yang), choid@kisti.re.kr (D.-H. Choi).
[1] These authors contributed equally to this work as the first author.

http://dx.doi.org/10.1016/j.ipl.2015.11.018
0020-0190/

also introduce a composite concept module if the operators are applied to its constituent concept modules. Each pattern of the constituent modules may be defined in terms of predefined rules. Unlike [1–3] detecting functional modules syntactically or structurally identical to users' needs based on exact matching, the concept module makes it possible to detect functional modules that are conceptually the same as well. Our method can also be adopted to concept-based detection or searching of chemical compounds [1] or multimedia information [4,5], as well as protein interactions [2,3].

2. Concept-based detection of functional modules

2.1. Interaction network

In the proposed method, an interaction network is represented by $N = < O_I, R >$, where O_I is a set of instance objects and R is a set of interaction relations (or simply relations) among them. Let $O_I(N)$ be a set of instance objects and $R(N)$ be a set of relations in the network. Proteins may be examples of instance objects whose properties can be referenced by the dot operator. For example, o.NAME is used to access its names and returns {“CDCrel1”, “SEPT5”, “Septin-5”, ...} for $o \in O_I(N)$. Similarly, to access its biological function, o.F is used and returns {“cytokinesis,” “regulation of exocytosis,” ...}.

Definition 1. Let each $r \in R(N)$ be a relation between instance objects. Then, r is defined as follows.

$$r = < o_1, o_2, type_{12} >,$$

where $type_{12} \in TYPE$ represents a type of relations between two instance objects $o_1, o_2 \in O_I(N)$ such as “bind,” “activate,” “regulate,” “decrease,” “increase,” etc.

Definition 2. Let $N = < O_I, R >$ be a network and M be all of the possible functional modules included in N. Then a functional module $m \in M$ is defined as follows.

$$m = < O_I, R >,$$

where $O_I(m) \subseteq O_I(N)$ and $R(m) \subseteq R(N)$.

2.2. Expression and evaluation of concept modules

The structure of a concept module is defined by an expression rule, or simply, a rule. It can be formulated by either a single triple or a set of triples together with related operators. To explain the rule, we first need to define a variable object. A variable object may be viewed as a template object which can be instantiated by a set of instance objects with the same names or functions.

2.2.1. Rule of concept modules

As a component of the triple, we start with a variable object as follows.

Definition 3. Let v be a variable object. Then a set of variable objects $O_V = 2^O_I$ is defined as

$$o_1, o_2 \in v$$

for $v \in O_V$ iff o_1.NAME $\cap$ o_2.NAME $\neq \Phi$ or o_1.F $\cap$ o_2.F $\neq \Phi$.

To express the relation between the variable objects, we now define a triple t.

Definition 4. Let $v_i, v_j \in O_V$ be variable objects and $type_{ij} \in TYPE$ be a type of relations respectively. Then a triple t is defined as follows.

$$t = < v_i, v_j, type_{ij} > .$$

Example 1. Suppose $v_1 = \{o_1\}$ and $v_2 = \{o_2, o_3\}$. Then, the triple $t = < v_1, v_2,$ “inactivate” $>$ is mapped to the relations, $r_1 = < o_1, o_2,$ “inactivate” $>$ and $r_2 = < o_1, o_3,$ “inactivate” $>$.

In the above example, $m_1 = < \{o_1, o_2\}, \{r_1\} >$ and $m_2 = < \{o_1, o_3\}, \{r_2\} >$ are the smallest form of functional modules which correspond to t. For this reason, a triple is considered to be a primitive concept module. In the following definition, we formalize an instance module of the triple.

Definition 5. Let $t = < v_1, v_2, type >$ be a triple and m be a functional module. Then we define

$$m \in \|t\| \quad \Leftrightarrow \quad \forall r = < o_1, o_2, type' > \in R(m),$$

where $o_1 \in v_1, o_2 \in v_2$ and $type' = type$.

When $m \in \|t\|$, we say m is an instance module of t, denoted by m_t.

The operators include two connection operators and one generalization operator. A connection operator is used to express a structural connection between two concept modules, whereas a generalization operator is employed to express a generalization relationship between them. For conceptual simplicity, “•” (arbitrariness) and “$*$” (association) are defined as the two connection operators and the “¦” notation is adopted as the generalization operator. “•” is used to express a concept module which includes two sub-modules unconditionally, while “$*$” is applied to express a concept module which concatenates two sub-modules if and only if their instances are directly connected. “$*_{\text{RESTRICTION}}$” is the more specific “$*$” operator with the constraint RESTRICTION = [DISTANCE(v_1, v_2) $<$ *length*]. The constraint specifies a concept module formed by concatenating other two concept modules if their instances are indirectly connected, keeping the path length between $o_i \in v_1$ and $o_j \in v_2$ is less than *length*.

Definition 6. Let OP $\in \{\bullet, *\}$ be the connection operators. Then each of the corresponding rules for its concept module c is defined as follows.

1) Let t be a triple, then $c \rightarrow t$ is a rule for c
2) Let c_1 and c_2 be concept modules. Then $c \rightarrow c_1$ OP c_2 is a rule for c