Hindawi
Mathematical Problems in Engineering
Volume 2020, Article ID 9234042, 12 pages
https://doi.org/10.1155/2020/9234042

Research Article

Identifying Key Classes Algorithm in Directed Weighted Class Interaction Network Based on the Structure Entropy Weighted LeaderRank

Wanchang Jiang and Ning Dai

School of Computer Science, Northeast Electric Power University, Jilin 132012, China

Correspondence should be addressed to Wanchang Jiang; jwchang84@163.com

Received 6 August 2020; Revised 16 November 2020; Accepted 24 November 2020; Published 10 December 2020

Academic Editor: Alessandro Tasora

Identifying key classes can help software maintainers quickly understand software systems. The existing key class recognition algorithms consider the weight of class interaction, but the weight mechanism is single or arbitrary. In this paper, the multitype weighting mechanism is considered, and the key classes are accurately identified by using four kinds of interaction. By abstracting the software system into the directed weighted class interaction network, a novel Structure Entropy Weighted LeaderRank of identifying key classes algorithm is proposed. First, considering multiple types and directions of interactions between every pair of classes, the directed weighted class interaction software network (DWCIS-Network) is built. Second, Class Entropy of each class is initialized by the software structural entropy in DWCIS-Network; the Structure Entropy Weighted LeaderRank applies the biased random walk process to iterate Class Entropy. Finally, the iteration is completed to obtain the Final Class Entropy (FCE) of each class as the importance score of each class, top-k classes are obtained, and key classes are identified. For two sets of experiments on Ant and JHotDraw, our approach effectively identifies key classes in class-level software networks for different top-k of classes, and the recall rates of our approach are the highest, 80% and 100%, respectively. From top-15% to top-5%, the precision of our approach is improved by 13.39%, which is the highest in comparison with the precisions of the other two classical approaches. Compared with the best performance of the two classical approaches, the RankingScore of our approach is improved by 16.51% in JHotDraw.

1. Introduction

As software grows in size and functionality, it becomes more difficult to understand and maintain. Understanding the functionality of the software accounts for the vast majority of the overall cost of software maintenance [1]. Software systems can also be represented as complex networks, usually termed as software networks, where nodes are software entities such as methods, classes, or packages, and edges represent interactions between entities [2].

Object-oriented (OO) programming is one of the most widely used programming paradigms for designing and implementing software systems [3]. Classes can be used to analyse and understand unfamiliar object-oriented software. Complex object-oriented software contains several closely related key classes that implement the main functions of the program. Therefore, it is a good choice to understand the software from the key classes. At present, there are many researches on key classes. From the functional perspective, Zaidman and Demeyer [4] considered the classes that have control functions to be key classes. When software dependency networks are used to model software systems, Şora and Chirila [5] believed that the key classes are those that manage other classes. From the point of view of influence, Ding et al. [6] considered key classes that are more likely to affect the structure and function in one network. It is more effective to analyse and understand software system starting with key classes [7]. Since classes are the basis of running the main functions of the system, it is important to propose effective key classes

identification approaches to reduce the cost of software understanding and maintenance.

In addition to defining key classes, how to identify key classes has also been studied by relevant researchers. To make the software easy to analyse, Valverde and Sole [8] proposed a class graph model based on the UML class diagram, where classes are represented as nodes and relationships between classes are represented as edges, ignoring the complexity of nodes, and the edges are unweighted. Chong and Lee [2] proposed a weighted complex network to analyse the maintenance ability and stability of object-oriented software. However, most of researchers assume that the relationship types between the nodes are the same when converting the source code to nodes and edges. In fact, software is relatively complex, and interactions between every pair of classes have directions and different types of interactions.

In the key classes identification, Wang et al. [7] used various complex network metrics to identify key classes from global and local perspectives. However, the weight of the edge only considers the frequencies of relationship between two classes. Meyer et al. [9] applied k-core decomposition to identify a core subset of vertices as potentially important classes. However, they ignored the fact that there may be multiple dependencies between classes. To deal with this problem, a generalized k-core decomposition method ICOOK has been proposed by Pan et al. [10]. Further, considering the influence of weights on key classes identification, Pan et al. [11] have proposed a weighted k-core decomposition approach to identify key classes. In addition, by defining various weights for the directed edges, Sora [12] applied the PageRank algorithm to identify the key classes. However, that could lead to suspending nodes and slowing down the sort. To improve the convergence and connectivity of the network, Lü et al. [13] proposed the standard LeaderRank by adding the ground node. LeaderRank is faster than PageRank at identifying influencers. For improving the speed of identifying influencers, a weighting mechanism is introduced to the LeaderRank, allowing nodes with more fans to get more scores from the ground node [14]. Zhang et al. [15] have proposed an improved weighted LeaderRank by taking clustering coefficient into account to depict the weight to identify influencers. Existing researches [16–18] show that structure entropy can be used to classify data based on ranking.

In order to identify key classes for helping software maintainers quickly understand software, the existing LeaderRank weighting mechanism is improved to take account of multiple types of class interactions and Structure Entropy Weighted LeaderRank is proposed for applying key classes identification. Then, we propose a novel identifying key classes algorithm based on the Structure Entropy Weighted LeaderRank (IKC-SEWL). First, class interactions in software source code are abstracted as a directed weighted class interaction software network (DWCIS-Network). Second, a Structure Entropy Weighted LeaderRank is proposed by introducing the biased random walk and class interaction weighting mechanism taking into consideration both the direction and weights of class interactions to obtain the class importance scores for key classes identification in DWCIS-Network. Finally, identifying key classes algorithm based on the Structure Entropy Weighted LeaderRank is proposed to identify key classes according to the obtained class importance scores. To evaluate the performance of our approach in identifying key classes of class-level software networks with multiple interaction types, two sets of experiments are conducted on two open-source software systems. When compared with three other key classes identification approaches, it is found that our approach has high identification precision.

The rest of the paper is organized as follows: Section 2 describes the directed weighted class interaction software network. Section 3 proposes the Structure Entropy Weighted LeaderRank. Based on that, Section 4 presents the key classes identification algorithm. Section 5 presents experiments to verify the effectiveness of our approach. Section 6 gives the final conclusion and looks forward to future work.

2. Directed Weighted Class Interaction Software Network

In order to identify key classes from the perspective of complex network, the structure topology information should be extracted from the source code of software. The extracted classes and class interactions are represented as nodes and edges, respectively. Class interactions in the object-oriented software system not only have directions but also contain different types. Inheritance, instantiation, return type, and method call between every pair of classes within the software system are selected and extracted, the directions and multiple interactions between every pair of classes are considered, and the directed weighted class interaction software network (DWCIS-Network) is built as DWCIS-Network $= (V, E, W, S)$. $V = \{v_i\}$ is the set of N classes in the network and v_i represents class i. $E = \{e_{ij}\}$ is the set of L directed weighted edges, in which $e_{ij} = (v_i, v_j)$ represents the class interaction from v_i to v_j $(v_i, v_j \in V)$. $W = \{w_{ij}\}$ is the weight set of directed edges, in which w_{ij} represents the weight of directed edge e_{ij}, that is, the sum of the multiplication of all interaction type weights and their frequencies between v_i and v_j. $S = \{s_i\}$ represents the set of software structural entropy of N classes, in which s_i represents software structural entropy of v_i $(v_i \in V)$.

According to the reality of Ant in [19, 20], there are four common types of class interactions, which account for more than 90% of all types. In our approach, according to Ioana Sora's idea of the relative proportions of class interactions rather than the actual values, different weights are empirically assigned to the four interaction types in DWCIS-Network, which are described as follows:

(1) Inheritance relation (inh): class i inherits from class j by "extends"; $w^{\text{inh}} = 3$

(2) Instantiation relation (ins): an instance of class j is created anywhere in the code that belongs to class i; $w^{\text{ins}} = 1$

(3) Return type relation (ret): one method of class i has the return type of class; $w^{\text{ret}} = 1$

(4) Method call relation (met): one method of class i calls at least one method of class j; $w^{\text{met}} = 2$

Our approach focuses on the relative proportion of class interactions as weights to build the set of weights $\{w^{\text{inh}}, w^{\text{ins}}, w^{\text{ret}}, w^{\text{met}}\}$, the interaction weight w^t is the weight of class interaction of type t, and f_{ij}^t is the frequency of interaction type t between v_i and v_j.

The weight w_{ij} can be computed as follows:

$$w_{ij} = \sum_{t=\text{inh,ins,ret,met}} f_{ij}^t \cdot w^t. \tag{1}$$

A simple example of constructing DWCIS-Network is shown in Figure 1, where Figure 1(a) is a code slice of Student class interaction and Figure 1(b) is its corresponding DWCIS-Network; and class "Task," class "UNStudent," and class "Student" are abstractly represented as nodes 1, 2, and 3, respectively. Taking class "Task" and class "UNStudent" as example, w_{12} represents the weight of directed edge "Task-UNStudent." The "Task" has a method "getStudent" that returns the "UNStudent." This method instantiates the "UNStudent" and calls the "study" method of the "UNStudent" once. Therefore, the weight w_{12} is 4 by using formula (1).

To reflect the significance of one class by the class interactions, the software structure entropy is calculated according to the relative number of edges owned by the class of the software network. Therefore, degree information of v_i is introduced to calculate the software structural entropy. In the following formulas, let N represent the number of identified software classes. Degree information in complex network including degree k_i, sum of indegree sum k^{in}, sum of outdegree sum k^{out}, and sum of degree sum k [21], respectively, is expressed as follows:

$$\begin{aligned} k_i &= k_i^{\text{in}} + k_i^{\text{out}}, \\ \text{sum}\, k^{\text{in}} &= \sum_{i=1}^{N} k_i^{\text{in}}, \\ \text{sum}\, k^{\text{out}} &= \sum_{i=1}^{N} k_i^{\text{out}}, \\ \text{sum}\, k &= \text{sum}\, k^{\text{in}} + \text{sum}\, k^{\text{out}}, \end{aligned} \tag{2}$$

where k_i^{in} represents the indegree of v_i and k_i^{out} represents the outdegree of v_i.

With the above information, the software structure entropy s_i can be obtained as follows:

$$s_i = -p_i \cdot \ln p_i, \tag{3}$$

$$p_i = \frac{k_i}{\text{sum}\, k} = \frac{k_i}{2 \cdot L}, \tag{4}$$

where p_i is proportional to the degree of v_i and $p_i \geq 0$, k_i represents degree of v_i, and L represents the number of directed edges in DWCIS-Network. The constraint condition of p_i is as follows:

$$\sum_{i=1}^{N} p_i = 1. \tag{5}$$

By using the above building process of DWCIS-Network, the software system Ant can be abstracted as class interaction network shown in Figure 2(a), where the ten key class nodes are marked red. Key classes will be identified and discussed in the experiment of Section 5. Class "Project" is abstracted node 25 and class "Task" is abstracted node 32, class nodes that interact with node 32 and node 25 are marked green, and the remaining class nodes are marked gray. The class interaction from node 32 to node 25 has 6 return types and 27 method calls, $w_{32\,25} = 60$, as is shown in Figure 2(b). Figure 2(b) represents local class interaction network of Ant based on the class interaction between node 32 and node 25. Similarly, other class interactions can be calculated according to the above calculation, and the weights of interactions between classes can be obtained. These weights are different and contain the multiplication of values of different interaction types and interaction frequency.

3. Structure Entropy Weighted LeaderRank

The Structure Entropy Weighted LeaderRank is proposed to obtain Final Class Entropy as the importance score of each class for key classes identification.

3.1. Initialization Stage. For the initialization stage, Class Entropy of each class is initialized by the software structural entropy in DWCIS-Network. Similar to the standard LeaderRank, the Structure Entropy Weighted LeaderRank adds a node named ground to DWCIS-Network, which makes the network strongly connected by bidirectional connection with each node. Unlike the standard random walk process in the standard LeaderRank, the Structure Entropy Weighted LeaderRank applies the biased random walk process, allowing more interacting classes to get more scores from the ground node. The initial Class Entropy $\text{CE}_i(0)$ of v_i can be represented by the software structure entropy of v_i (s_i) at the initial stage of the Structure Entropy Weighted LeaderRank. Let $N+1$ represent the position of the ground node relative to all class nodes and $\text{CE}_{N+1}(0)$ represents the initial Class Entropy of the ground node as follows:

$$\text{CE}_i(0) = s_i, \tag{6}$$

$$\text{CE}_{N+1}(0) = 0. \tag{7}$$

The edge weights from ground node to v_i are assigned by s_i, $w_{N+1\,i} = s_i$, and the edge weights from v_i to ground are assigned by 1, and $w_{i\,N+1} = 1$. The weight w_{ij} of the directed edge from v_i to v_j should be initialized, which is as follows:

$$w_{ij} = \frac{w_{ij}}{\text{sum}\, w_i^{\text{out}}}, \tag{8}$$

where sum w_i^{out} represents the sum of weight of directed edge e_{ij} [22], that is, the outstrength of v_i, and it is expressed as follows:

$$\text{sum } w_i^{\text{out}} = \sum_{j=1}^{N+1} w_{ij}. \tag{9}$$

3.2. Iteration Stage. For the iteration stage, the Class Entropy CE_i is iterated through a biased random walk process by using weights of directed edges and the betweenness of class i. The Class Entropy CE_i is proportional to the betweenness of class i. $\text{CE}_i(t+1)$ is the Class Entropy of v_i at $(t+1)$-th iteration. As the betweenness is defined based on the shortest path of the network [23], it makes up the shortcoming that Class Entropy only considers local characteristics. It is used as the regulatory factor of the iteration of CE_i and CE_i is updated by the following rules:

$$\text{CE}_i(t+1) = \left[\sum_{j=1}^{N+1} w_{ij} \cdot \text{CE}_i(t)\right] \cdot bc_i, \tag{10}$$

where $\text{CE}_i(t)$ is the Class Entropy of v_i at t-th iteration, w_{ij} represents the weight of directed edge from v_i to v_j, and bc_i represents the weighted betweenness of v_i. The weighted betweenness is computed using the weighted shortest paths that not only consider the number of edges necessary to travel between nodes but also consider the weight attached to the links in complex network [24]. The shortest path between two nodes is the minimum number of edges to travel from a node to the other. The weighted shortest paths not only consider the number of edges necessary to travel between nodes but also consider the weight attached to the edges. The definition of bc_i is expressed as follows:

$$bc_i = \sum_{j}^{N+1} \sum_{l}^{N+1} \frac{g_i^{jl}}{g^{jl}}, \quad j \neq l \neq i, \tag{11}$$

where g^{jl} is the total weight of the weighted shortest path from v_j to v_l; g_i^{jl} is the weight of the weighted shortest path through v_i from v_j to v_l.

3.3. Iteration Completion Stage. When the difference in the Class Entropy sum of two iterations is less than 1/1000 of the Class Entropy sum of the previous iteration, the Class Entropy is considered to have reached a stable state, and the iteration of Class Entropy is completed. The constraint condition of the stable state iteration times t_s is as follows:

$$\sum_{i=1}^{N} \text{CE}_i(t_s) - \sum_{i=1}^{N} \text{CE}_i(t_s - 1) < \frac{1}{1000} \sum_{i=1}^{N} \text{CE}_i(t_s - 1). \tag{12}$$

As the iteration of CE_i reaches a stable state, like WLeaderRank, the weight of a ground node is eventually split equally among the nodes in the network, and the ground node is then removed from the network; the Final Class Entropy FCE_i of v_i is obtained by the following formula:

$$\text{FCE}_i = \text{CE}_i(t_s) + \frac{\text{CE}_{N+1}(t_s)}{N}, \tag{13}$$

$$\text{FCE}_{N+1} = 0, \tag{14}$$

where $\text{CE}_{N+1}(t_s)$ represents the Final Class Entropy of the ground node at stable state, FCE_{N+1} represents importance score of the ground node, and FCE_i represents importance score of class i for key classes identification.

4. Identifying Key Classes Algorithm Based on the Structure Entropy Weighted LeaderRank

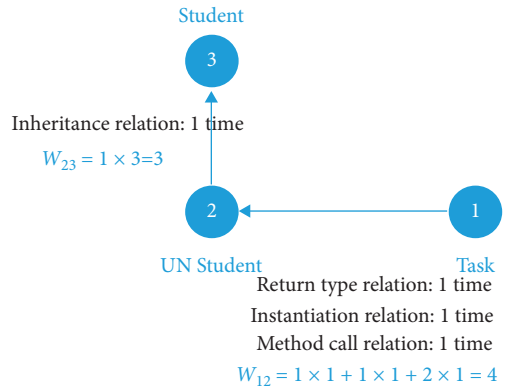
Identifying key classes algorithm based on the Structure Entropy Weighted LeaderRank (IKC-SEWL) is proposed to identify key classes in DWCIS-Network. By using Final Class Entropy (FCE) in the Structure Entropy Weighted LeaderRank, the importance of class is measured, and, as a result, key classes in DWCIS-Network can be identified. The overview of IKC-SEWL is shown in Figure 3.

The detailed procedure of the algorithm is described in Algorithm 1.

The algorithm IKC-SEWL is divided into four stages: constructing DWCIS-Network, initializing and iterating the Structure Entropy Weighted LeaderRank, and identifying key classes. In the stage of constructing DWCIS-Network, from step 1 to step 6, the source code is abstracted and software structural entropy and the weights of edges are calculated to construct DWCIS-Network. The initialization stage includes steps 7 to 11; the Class Entropy of each class and the weights of directed edges between classes are initialized in the Structure Entropy Weighted LeaderRank by using the information of DWCIS-Network. In the iteration stage, including steps 12 to 17, the Class Entropy (CE) is proportional to the betweenness for each class and the Class Entropy is iterated in the biased random walk process. When the number of iterations satisfies certain constraints, the Final Class Entropy (FCE) is obtained. In the key classes identification stage, step 18, by using corresponding scores in the set of $\{\text{FCE}_i\}$, classes in DWCIS-Network are arranged in descending order. The top-k of ordered classes are obtained as the candidate set of key classes, and key classes can be finally identified in terms of key classes in the design document. Because the weighted matrix W needs to be traversed, the time complexity is $O(n^2)$, and the remaining operations only nest one layer for loop, so the maximum time complexity is $O(n^2)$, and n is the number of class nodes.

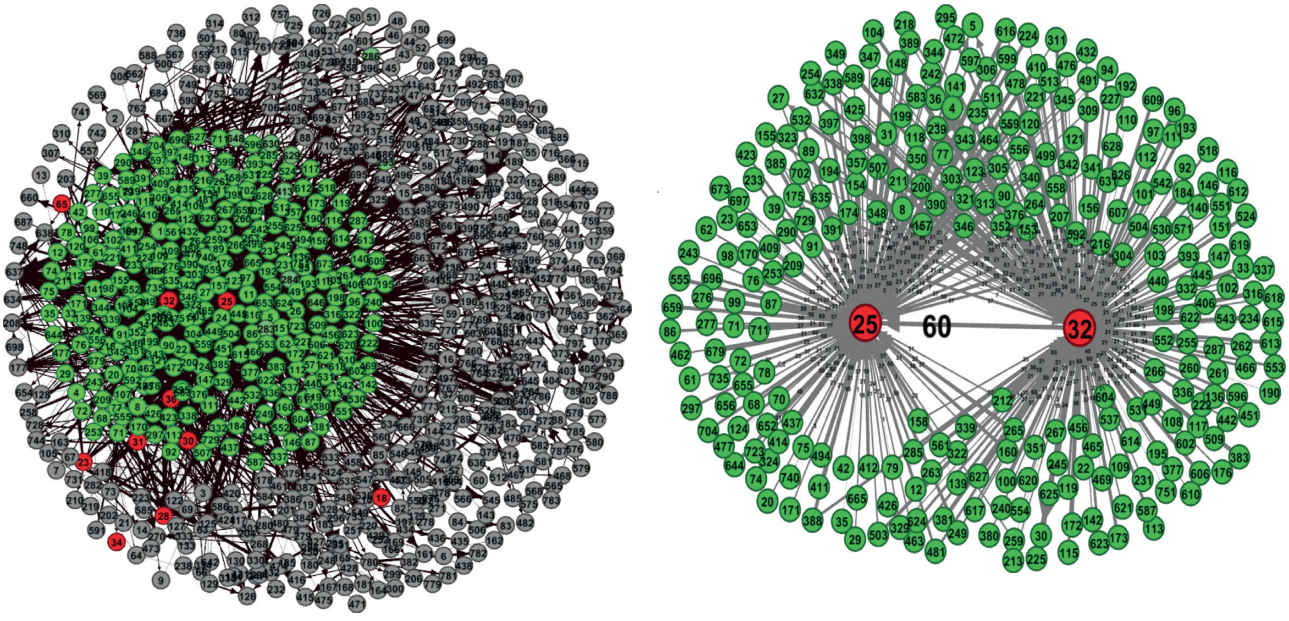
```
public class Student{
    public void study(){
        System.out.println("Student study! ");
    }
}

public class UNStudent extends Student{
    public void study(){
        System.out.println("UNStudent study");
    }
}

public class Task{
    public UNStudent getStudent(){
        UNStudent unstudent =new UNStudent();
        unstudent.study();
        return unstudent;
    }
}
```

(a)



(b)

Figure 1: Student class interaction code and its corresponding DWCIS-Network. (a) The code slice of Student class interaction. (b) The DWCIS-Network.



(a) (b)

Figure 2: DWCIS-Network of Ant. (a) Class interaction network of Ant. (b) Local class interaction network of Ant based on the class interaction between node 32 and node 25.

5. Experiment

Two sets of experiments are designed by using software systems Ant and JHotDraw, respectively. By comparison with existing works, including software key class identification model (SKCI) [7], identifying key class candidates in OO software using generalized k-core decomposition (ICOOK) [10], and Weighted LeaderRank (WLeaderRank) [14], which are classical approaches in the last two, the effectiveness of IKC-SEWL in identifying key classes is verified. Experiments are done on a PC at Inter(R) Core (TM) i7-9750H CPU @ 2.6 GHz with 8 GB of RAM.

5.1. Experiment Purpose. The experiments have two purposes as follows:

(i) Purpose 1: to verify whether the algorithm IKC-SEWL is efficient at identifying key classes in DWCIS-Network with the threshold of top-15%

(ii) Purpose 2: to verify whether the algorithm IKC-SEWL is efficient to the case with different top-k

For these purposes, the experimental comparison and analysis are evaluated by the four following evaluation criteria:

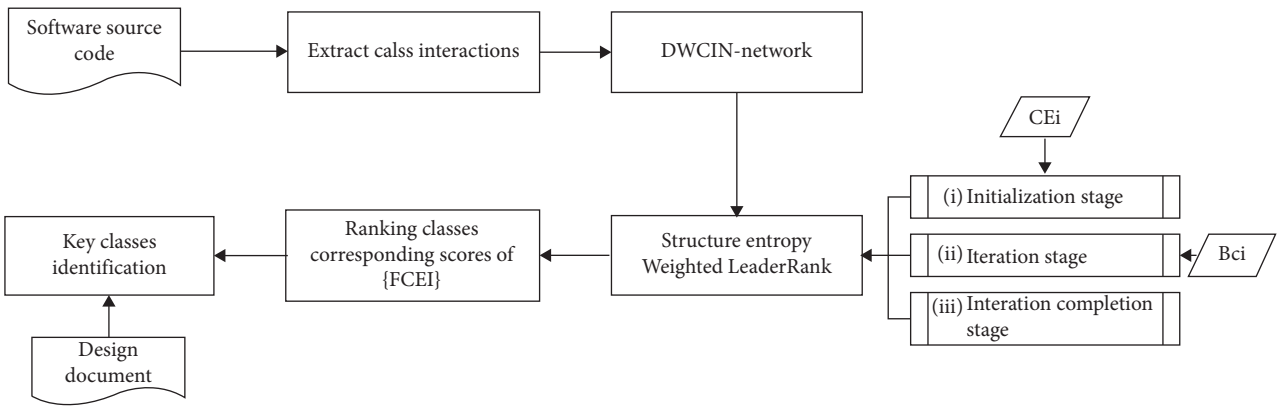


Figure 3: The overview of IKC-SEWL.

(i) Precision [4]: the ratio of the number of key classes obtained by a specific approach relative to the total number of top-k classes selected

(ii) Recall: the ratio of the number of key classes obtained by a specific approach to the number of all known key classes

(iii) RankingScore [25]: the average position for key classes positions found by a particular approach

(iv) Time: the time consumption from the software network construction to key classes identification

Precision and Recall are used to evaluate whether a particular approach is capable of identifying key classes in a software system and determine which approach performs best. RankingScore is used to measure the average position of key classes in the ranked list of classes obtained by a particular approach. Obviously, a good approach is expected to give high recommendations to true key classes, thus leading to high ranking score values.

5.2. Experiment Objects. Two different types of software systems, Ant and JHotDraw, are used as experiment objects. Ant is a Java library and command-line tool designed for building Java applications. JHotDraw is a Java graphics framework for two-dimensional graphics editors; there are many interactions between every pair of classes that can be used for key classes analysis. The characteristics are as follows:

(i) They have design documents containing key classes to verify the effectiveness

(ii) There are related researches of identifying key classes for comparison

5.3. Experimental Results and Analysis. By using class interactions in Section 2, DWCIS-Network models are constructed for Ant and JHotDraw, respectively. Table 1 shows an overview of Ant and JHotDraw under four approaches (IKC-SEWL, WLeaderRank, ICOOK, and SKCI), the column Version is the version of software. Ant1.6.1 is identified by four approaches. JHotDraw5.1 is identified by the SKCI, and JHotDraw6.0b.1 is identified by the other three approaches. The column Node is the number of classes and the column Edge is the number of interactions between every pair of classes.

5.3.1. Effectiveness of Key Classes Identification with Top-15% Classes. In related studies [4, 5, 10], the recommended threshold value of key classes is top-15%. Therefore, we use the threshold of top-15% to compare the performance of the algorithm IKC-SEWL with other key classes identification approaches. There are ten key classes and nine key classes from the design document in the core of Ant and JHotDraw, respectively, which are used as a benchmark for key classes identification approaches. Recall and Precision and RankingScore and Time are used to measure the effectiveness of approaches.

Tables 2 and 3 show the results of top-15% key classes identification of different approaches applied to Ant and JHotDraw. The first column "Order no." represents the order of key classes. The second column contains key classes extracted from the design document. For example, key classes in Table 2 can be successively abstracted as the nodes marked red in DWCIS-Network of Figure 2(a), and the Node ID column lists the Node ID of each class. The other three columns represent identified key classes by three approaches, IKC-SEWL, ICOOK, and WLeaderRank, respectively. The classes that are left behind in the ranking are "TaskContainer" and "ElementHandler." By examining the code, class "ElementHandler" is an inner class contained in class "ProjectHelper." Class "ProjectHelper" ranks higher, at position 12. It is rather unusual that the benchmark mentions an inner class as a key class, as the public classes have bigger architectural impact. "TaskContainer" is an interface that is not an actual class and defines objects that can contain tasks, so it is not identified as a key class and marked as "*N*" in "Identified" column. Its most important implementing class is "Target," a key class that is highly ranked by our approach. The identified position of class "ElementHandler" under IKC-SEWL is 34 for Ant, so it can be identified by IKC-SEWL and marked as "*Y*." Since the position at 220 cannot be identified by ICOOK, it is marked as "*N*."

For Ant, take several classes in Table 2 as an example: the class "Project" is instantiated whenever Ant starts and, with

Input: software source code.
Output: key classes in DWCIS-Network.
(1) **for** each class interaction from class i to class j **do** 2–5 /* DWCIS-Network construction */
(2) Extract class i, class j as v_i, v_j and put them into set V;
(3) Extract the class interaction from class i to class j as e_{ij} and put it set E;
(4) w_{ij} is calculated for e_{ij} by formula (1), put it into set W;
(5) s_i and s_j are calculated by formulas (3) and (4) and put them into set S;
(6) With 1–5, DWCIS-Network (V, E, W, S) is obtained;
(7) **for** each v_i in V **do** 8/* Initialization stage */
(8) CE_i is initialized with s_i by formula (6);
(9) CE_{N+1} is initialized with 0 by formula (7);
(10) **for** each w_{ij} in $N + 1$ dimensional W **do** 11
(11) w_{ij} is initialized by formulas (8) and (9);
(12) **while** the constraint condition of formula (12) is not true **do** 13–15 /* Iteration stage */
(13) **for** each v_i in V of $N + 1$ dimensional **do** 14, 15
(14) bc_i is calculated by formula (11);
(15) CE_i is updated by formula (10);
(16) **for** each v_i in V do 17 /* Iteration completed */
(17) FCE_i is calculated by formula (13);
(18) All classes are ranked in descending order by using corresponding scores in the set of $\{\mathrm{FCE}_i\}$, top-k classes are obtained and key classes are identified;

Algorithm 1: IKC-SEWL.

Table 1: The overview of Ant and JHotDraw under four approaches.

System	IKC-SEWL			WLeaderRank			ICOOK			SKCI		
	Version	Node	Edge	Version	Node	Edge	Version	Node	Edge	Version	Node	Edge
Ant1.6.1	1.6.1	797	2518	1.6.1	797	2518	1.6.1	900	2672	1.6.1	403	—
JHotDraw	6.0b.1	517	1591	6.0b.1	517	1591	6.0b.1	544	—	5.1	155	—

The symbol "—" indicates that the data is not mentioned in the paper.

Table 2: The top-15% key classes identification in Ant under three approaches.

Order no.	Key classes	Node ID	IKC-SEWL		ICOOK		WLeaderRank	
			Identified	Position	Identified	Position	Identified	Position
1	Project	25	Y	4	Y	1	Y	5
2	Task	32	Y	8	Y	3	Y	9
3	ProjectHelper	28	Y	12	Y	55	Y	16
4	RuntimeConfigurable	30	Y	23	Y	51	Y	17
5	UnknownElement	36	Y	15	Y	36	Y	18
6	IntrospectionHelper	18	Y	19	Y	43	Y	21
7	Target	31	Y	20	Y	19	Y	28
8	Main	23	N	142	N	208	N	271
9	TaskContainer	34	N	142	Y	101	N	308
10	ElementHandler	65	Y	34	N	220	N	447
Recall (%)			80		80		70	
Precision (%)			6.69		5.90		5.86	
RankingScore			41.9		73.7		114	
Time (s)			13.763		48.078		12.277	

the help of helper classes, the class "Project" instance parses the build.xml file. The class "Target" represents the targets specified in the build.xml file. Once parsing finishes, the build model consists of a project, containing multiple targets. As a container of tasks, the class "Target" is represented by specializations of the class "Task." Each task in Ant has a reference to its "RuntimeConfigurable" instance. Prior to the task being executed, it would need to be configured from its "RuntimeConfigurable" instance. The "Main" and "ElementHandler" are not identified by ICOOK because there are fewer classes attached to them. Unlike that, "Main" failed to be identified by IKC-SEWL because "Main" has low

betweenness value, resulting in a significant decline in the influence of this class in DWCIS-Network. In terms of Time, WLeaderRank performs better than IKC-SEWL, because WLeaderRank fails to consider the effect of betweenness of classes during iteration.

For Ant, while the top-15% of ICOOK has 135 candidate key classes and the top-15% of IKC-SEWL only contains 120 candidate key classes, these two approaches have the same number in identifying key classes. Therefore, IKC-SEWL has a higher precision than ICOOK in identifying key classes. In terms of RankingScore and Time, IKC-SEWL has a higher average rank than ICOOK and WLeaderRank and uses less time to identify key classes than ICOOK.

As shown in Table 3, IKC-SEWL selects top-15% of 517 classes; nine key classes are all identified according to the positions obtained by arranging the scores of classes. Precision can be calculated as $9/(517 * 15\%)$ and Recall can be calculated as $(9/9) * 100\%$. The results of identifying key classes in JHotDraw are obtained, and the performance parameters of IKC-SEWL are Recall, 100%, Precision, 11.61%, RankingScore, 10.11, and Time, 14.435 s, all of which are better than those of ICOOK. Except for time, IKC-SEWL's other performance parameters are all better than WLeadeRank's. Among them, both ICOOK and WLeaderRank have the same Recall. However, in terms of RankingScore, key classes average position of IKC-SEWL is 2 and 14.557 higher than that of ICOOK and WLeaderRank. Therefore, IKC-SEWL has better performance than ICOOK and WLeaderRank in RankingScore.

As shown in Table 2, Precision of IKC-SEWL for Ant is 6.69%, which is 0.79% and 0.83% higher than that of ICOOK and WLeaderRank. From Table 3, Precision of IKC-SEWL for JHotDraw is 11.61%, which is 0.58% and 1.29% higher than that of ICOOK and WLeaderRank.

As shown in Figures 4 and 5, each approach is applied to identify the key classes of Ant and JHotDraw, and the position of the key classes varies. Average position of identified key classes of IKC-SEWL, the RankingScore, is higher than that of ICOOK and WLeaderRank. In Figure 4, on the horizontal axis, the "order of the key class" corresponds to the order of the ten key classes listed in the column "Order no." in Table 2. The same can be said for Figure 5.

For Ant, the number of candidate key classes of WLeaderRank and IKC-SEWL is 120, while the number of candidate key classes identified by ICOOK is 135. From Figure 4, three nodes, far away from 120, are not identified by WLeaderRank. IKC-SEWL can identify the key classes with the position no more than 141. The positions for order no. 8 and order no. 9 are 142 and 142, respectively. They are all higher than 141, because, in the software source code for the two classes "Main" and "TaskContainer," there are only a few interactions with other classes.

ICOOK and IKC-SEWL all have identified eight key classes, but the average position of key classes identified by IKC-SEWL is 43.15% higher than that identified by ICOOK, so IKC-SEWL has better performance than ICOOK. From Figure 5, IKC-SEWL and ICOOK identify all key classes; that is, the positions of all key classes are at top-15%; furthermore, the RankingScore of IKC-SEWL is 16.51% higher than that of ICOOK. So, IKC-SEWL can effectively identify key classes in top-15% classes.

5.3.2. Effectiveness of Key Classes Identification for Different Top-K. In addition to the effectiveness of key classes identification by IKC-SEWL in top-15% classes, two set of experiments are conducted to verify the effectiveness of key classes identification by IKC-SEWL in top-10% classes. From Table 4, in terms of Precision, SKCI has a precision rate of 18% and identifies seven key classes which are as many as those of WLeaderRank and ICOOK but one less than IKC-SEWL. Among IKC-SEWL, WLeaderRank, and ICOOK, Precision of IKC-SEWL is the highest, 10.04%. From the view of Recall, IKC-SEWL identifies eight key classes, and the Recall rate is up to 80%. However, in top-10% classes (90 classes), ICOOK cannot identify "TaskContainer," and the Recall rate of ICOOK is reduced to 70%. Although WLeaderRank has the shortest time, it sacrifices performance of Recall and Precision. In addition, IKC-SEWL takes 71.37% less time to identify key classes than ICOOK and 77.06% less time than SKCI, which greatly improves the speed of identifying key classes.

As shown in Table 5, SKCI does not mention the RankingScore of key classes for JHotDraw; four approaches are compared under three perspectives. The Recall rates of SKCI, WLeaderRank, ICOOK, and IKC-SEWL are 56%, 77.8%, 100%, and 100%, respectively, indicating that ICOOK and IKC-SEWL can identify all key classes in JHotDraw. Among IKC-SEWL, WLeaderRank, and ICOOK, Precision of IKC-SEWL is the highest, 17.41%. Precision of identifying key classes of SKCI is 33% higher than that of other approaches, but the identified candidate set is only 155 classes, less than the 544 classes of ICOOK and 517 classes of IKC-SEWL and WLeaderRank. As a result, the identified candidate classes by SKCI and Recall of identifying key classes dropped significantly. In addition, SKCI takes 30 s and only identifies 155 classes, and IKC-SEWL takes less time (14.435s) and more key classes are identified. WLeaderRank ignores the influence of betweenness on the importance of nodes and simplified the identification process of key classes, so it improves time performance at the cost of reducing Recall and Precision.

For top-10% classes, it is still likely to be a larger candidate set range and to further verify that our approach is still effective for identifying key classes on a smaller candidate set range. The top-5% classes will be selected as the candidate set to determine whether key classes are identified according to the class rank identified by each approach. Because SKCI does not provide the position information for identifying key classes, the comparison is only made in the other three approaches.

Ant is used as an example, and it can be known that, in the top-5% of Ant, IKC-SEWL, ICOOK, and WLeaderRank can identify 40, 45, and 40 classes, respectively. From Tables 2 and 3, we can see the positions of each key class identified by the three approaches. We found that IKC-SEWL, ICOOK, and WLeaderRank identify 8, 5, and 7 key

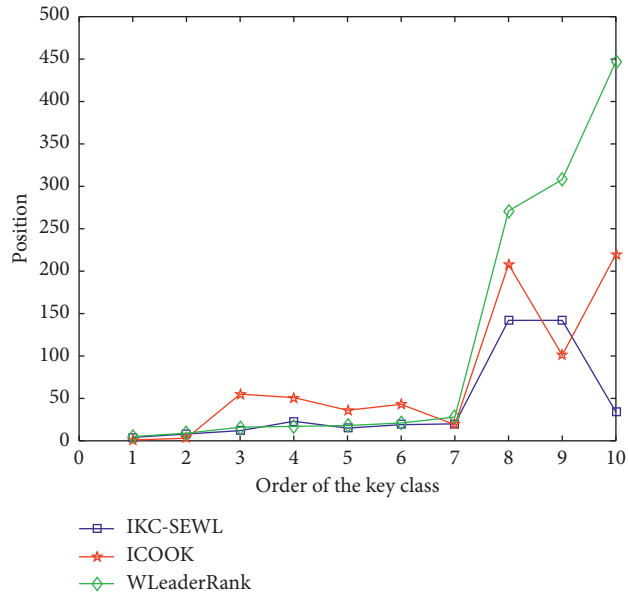


Figure 4: Key classes nodes ranking for each approach applied to Ant.

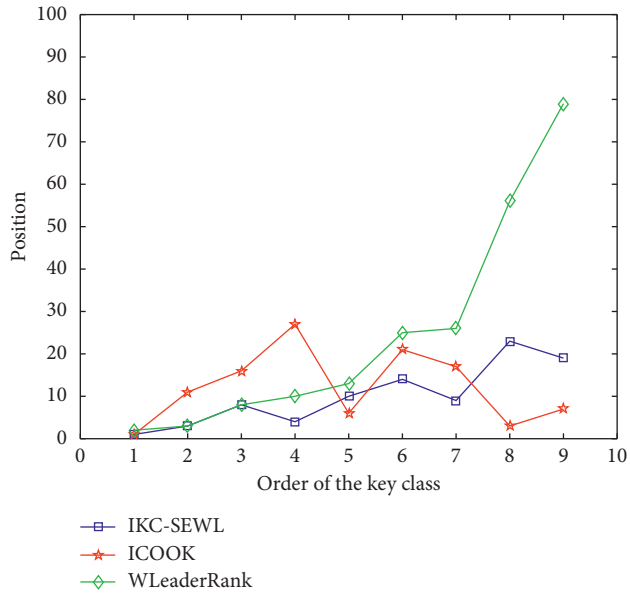


Figure 5: Key classes nodes ranking for each approach applied to JHotDraw.

classes, respectively. Therefore, the Precision rate of IKC-SEWL is 20.08% (8/(797 ∗ 5%)), and those of ICOOK and WLeaderRank are 11.11% and 17.57%, respectively. For different top-k, the Precision of three approaches in identifying key classes in Ant is shown in Table 6.

Figure 6 represents the variation trend of Precision of identifying key classes in Ant when varying the top-k between 5% and 15%. From Figure 6, with the gradual decrease of k of top-k, the Precision of key class identification is increasing, which indicates that positions of key classes are at higher position in the identification results of various approaches. Because the positions of class "ProjectHelper" and class "RuntimeConfigurable" are 55 and 51, respectively, higher than the top-5% (45 classes) identified by ICOOK, these two classes cannot be identified, so the Precision of ICOOK is the lowest among the top-5% key classes identified. Because WLeaderRank only identified 7 key classes at the top-5%, its Precision is lower than that of IKC-SEWL. From top-15% to top-5%, the Precision of our approach is improved by 13.39%, which is highest in comparison with the Precision of the other two classical approaches. This shows that our approach can still efficiently identify key classes in a small range of candidate set.

For the overall identification performance of top-10% key classes on Ant and JHotDraw and the Precision from top-15% to top-5%, the performance of IKC-SEWL still

Table 3: The top-15% key classes identification in JHotDraw under three approaches.

Order no.	Key classes	IKC-SEWL		ICOOK		WLeaderRank	
		Identified	Position	Identified	Position	Identified	Position
1	Figure	Y	1	Y	1	Y	2
2	DrawingView	Y	3	Y	11	Y	3
3	DrawingEditor	Y	8	Y	16	Y	8
4	Handle	Y	4	Y	27	Y	10
5	Drawing	Y	10	Y	6	Y	13
6	DrawApplication	Y	14	Y	21	Y	25
7	Tool	Y	9	Y	17	Y	26
8	CompositeFigure	Y	23	Y	3	Y	56
9	StandardDrawingView	Y	19	Y	7	N	79
Recall (%)		100		100		88.9	
Precision (%)		11.61		11.03		10.32	
RankingScore		10.11		12.11		24.667	
Time (s)		14.435		47.867		11.581	

Table 4: Top-10% key classes identification in Ant under four approaches.

Order no.	Key classes	IKC-SEWL		ICOOK		SKCI		WLeaderRank	
		Identified	Position	Identified	Position	Identified	Position	Identified	Position
1	Project	Y	4	Y	1	Y	—	Y	5
2	Task	Y	8	Y	3	Y	—	Y	9
3	ProjectHelper	Y	12	Y	55	Y	—	Y	16
4	RuntimeConfigurable	Y	23	Y	51	Y	—	Y	17
5	UnknownElement	Y	15	Y	36	Y	—	Y	18
6	IntrospectionHelper	Y	19	Y	43	Y	—	Y	21
7	Target	Y	20	Y	19	Y	—	Y	28
8	Main	N	142	N	208	N	—	N	271
9	TaskContainer	N	142	N	101	-	—	N	308
10	ElementHandler	Y	34	N	220	N	—	N	447
Recall (%)		80		70		77		70	
Precision (%)		10.04		7.78		18		8.78	
RankingScore		41.9		73.7		—		114	
Time (s)		13.763		48.078		60.000		12.277	

Table 5: Top-10% key classes identification in JHotDraw under four approaches.

Order no.	Key classes	IKC-SEWL		ICOOK		SKCI		WLeaderRank	
		Identified	Position	Identified	Position	Identified	Position	Identified	Position
1	Figure	Y	1	Y	1	—	—	Y	2
2	DrawingView	Y	3	Y	11	—	—	Y	3
3	DrawingEditor	Y	8	Y	16	—	—	Y	8
4	Handle	Y	4	Y	27	—	—	Y	10
5	Drawing	Y	10	Y	6	—	—	Y	13
6	DrawApplication	Y	14	Y	21	—	—	Y	25
7	Tool	Y	9	Y	17	—	—	Y	26
8	CompositeFigure	Y	23	Y	3	—	—	N	56
9	StandardDrawingView	Y	19	Y	7	—	—	N	79
Recall (%)		100		100		56		77.8	
Precision (%)		17.41		16.54		33		13.54	
RankingScore		10.11		12.11		—		24.667	
Time (s)		14.435		47.867		30.000		11.581	

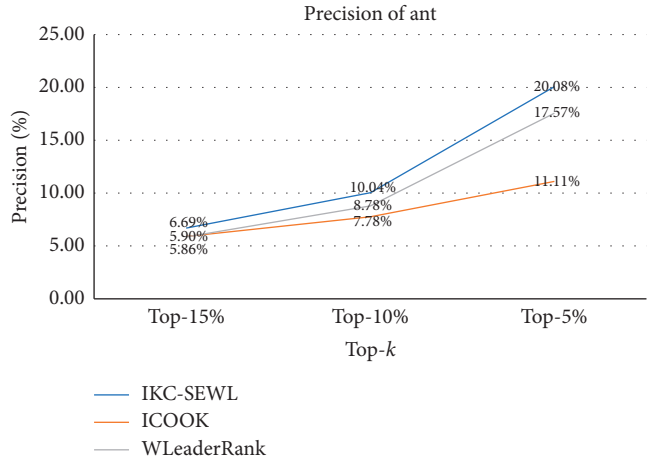


Figure 6: The trend of Precision of identifying key classes in Ant for different top-k.

Table 6: The Precision of three approaches for different top-k in Ant.

	Top-15 (%)	Top-10 (%)	Top-5 (%)
IKC-SEWL	6.69	10.04	20.08
ICOOK	5.90	7.78	11.11
WLeaderRank	5.86	8.78	17.57

remains effective when our approach identifies key classes in a small range of candidate sets of key classes.

For the positions identified by key classes, the average position RankingScore obtained by IKC-SEWL is 43.15% and 16.51% higher than that obtained by ICOOK for Ant and JHotDraw, respectively, which indicates that the key classes identified by our method may be obtained in a small top-k, because average positions of key classes are high. By analysing from top-15% to top-5% of classes as the candidate set of key classes, compared with the average Precision of ICOOK and WLeaderRank, we can conclude that our approach identifies key classes with the highest precision for Ant. Compared with the Recall rates of ICOOK, WLeaderRank, and SKCI, IKC-SEWL has the highest Recall rate of 80% for Ant and 100% for JHotDraw. In terms of Time, IKC-SEWL reduces the average time of the other three approaches by 65.69% and 51.58% for Ant and JHotDraw, respectively.

ICOOK assigns weights either 1 or 10 for each classes coupling, IKC-SEWL selects four kinds of class interaction types, and assigns different weights, which makes the identified key classes have higher position than ICOOK. When comparing it with WLeaderRank and SKCI, we find that the latter two only consider the number of interactions as the weight when calculating weights, because the software is complex and does not consider the influence of the type of class interactions on the class measurement. IKC-SEWL considered the weighted intermedium in the iterative process and found in the experimental process that the algorithm iteration would quickly converge, which improved the recognition speed. Because of the introduction of weighted intermedium, it is more consistent with the identification of key classes in the multitype interactive software network.

Therefore, IKC-SEWL effectively identifies key classes in class-level software networks with multiple types of interactions for candidate sets of different sizes. With high Precision, it can help software maintainers to quickly understand the software and reduce the cost of software maintenance.

6. Conclusion

In this paper, we propose a novel identifying key classes algorithm IKC-SEWL to identify key classes for helping software maintainers quickly understand software systems. The software system is firstly abstracted into a directed weighted class interaction network. Then the Structure Entropy Weighted LeaderRank is proposed to initialize Class Entropy of each class with the software structural entropy in DWCIS-Network and applies the biased random walk process to iterate Class Entropy. When the iteration is completed, the Final Class Entropy of each class is obtained and used as the importance score of each class. Finally, according to importance scores of classes, top-k classes are obtained and key classes are identified.

Java open software systems are used to evaluate the effectiveness of the key classes identification algorithm proposed in this paper. Results of experiments show that the Precision and Recall of our approach are higher than the related approaches, and the RankingScore ranks higher. Based on the above, our approach can identify key classes and performs well in aspects.

In the future, the scalability and universality of our approach will be verified in data sets of large-scale and package software networks. Except for mining key classes in the class-level software network, identification of key

components in software networks of different granularity including function network and package network will be verified by further experiments.

Data Availability

The data used to support the findings of this study have not been made available because the ownership of the tools that process the software source code is not made public.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported in part by the project of the National Natural Science Foundation of China under Grant 61572420, in part by Research Project of the Education Department of Jilin Province under Grant JJKH20190706KJ, and in part by Science and Technology Innovation Development Program of Jilin Province under Grant 20190104140.

References

[1] F. Fittkau, A. Krause, and W. Hasselbring, "Software landscape and application visualization for system comprehension with ExplorViz," *Information and Software Technology*, vol. 87, pp. 259–277, 2017.

[2] C. Y. Chong and S. P. Lee, "Analyzing maintainability and reliability of object-oriented software using weighted complex network," *Journal of Systems and Software*, vol. 110, pp. 28–53, 2015.

[3] J. Saraiva, "A roadmap for software maintainability measurement," in *Proceedings of the 2013 35th International Conference on Software Engineering (ICSE)*, pp. 1453–1455, San Francisco, CA, USA, May 2013.

[4] A. Zaidman and S. Demeyer, "Automatic identification of key classes in a software system using webmining techniques," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 20, no. 6, pp. 387–417, 2008.

[5] I. Şora and C. B. Chirila, "Finding key classes in object-oriented software systems by techniques based on static analysis," *Information and Software Technology*, vol. 116, pp. 1–15, Article ID 106176, 2019.

[6] Y. Ding, B. Li, and P. He, "An improved approach to identifying key classes in weighted software network," *Mathematical Problems in Engineering*, vol. 2016, Article ID 3858637, 9 pages, 2016.

[7] J. Wang, J. Ai, Y. Yang, and W. Su, "Identifying key classes of object-oriented software based on software complex network," in *Proceedings of the 2017 2nd International Conference on System Reliability and Safety (ICSRS)*, pp. 444–449, Milan, Italy, December 2017.

[8] S. Valverde and R. Sole, "Hierarchical small-worlds in software architecture," *Dynamics of Continuous Discrete and Impulsive Systems: Series B; Applications and Algorithms*, vol. 14, pp. 1–6, 2007.

[9] P. Meyer, H. Siy, and S. Bhowmick, "Identifying important classes of large software systems through K-core decomposition," *Advances in Complex Systems*, vol. 17, no. 7, pp. 25–32, 2015.

[10] W. Pan, B. Song, K. Li, and K. Zhang, "Identifying key classes in object-oriented software using generalized k-core decomposition," *Future Generation Computer Systems*, vol. 81, pp. 188–202, 2018.

[11] W. Pan, B. Li, J. Liu, Y. Ma, and B. Hu, "Analyzing the structure of Java software systems by weighted *K*-core decomposition," *Future Generation Computer Systems*, vol. 83, pp. 431–444, 2018.

[12] I. Sora, "A PageRank based recommender system for identifying key classes in software systems," in *Proceedings of the 2015 IEEE 10th Jubilee International Symposium on Applied Computational Intelligence and Informatics*, pp. 495–500, Timisoara, Romania, May 2015.

[13] L. Lü, Y. C. Zhang, C. H. Yeung, and T. Zhou, "Leaders in social networks, the delicious case," *PLoS One*, vol. 6, no. 6, Article ID e21202, 2011.

[14] Q. Li, T. Zhou, L. Lü, and D. Chen, "Identifying influential spreaders by weighted LeaderRank," *Physica A: Statistical Mechanics and Its Applications*, vol. 404, no. 15, pp. 47–55, 2014.

[15] Z. Zhang, G. Jiang, Y. Song, L. Xia, and Q. Chen, "An improved weighted LeaderRank algorithm for identifying influential spreaders in complex networks," in *Proceedings of the 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, pp. 748–751, Guangzhou, China, July 2017.

[16] F. A. N. Palmieri and D. Ciuonzo, "Objective priors from maximum entropy in data classification," *Information Fusion*, vol. 14, no. 2, pp. 186–198, 2013.

[17] F. Zhao, L. Jiao, H. Liu, X. Gao, and M. Gong, "Spectral clustering with eigenvector selection based on entropy ranking," *Neurocomputing*, vol. 73, no. 10-12, pp. 1704–1717, 2010.

[18] F. Palmieri and D. Ciuonzo, "Data fusion with entropic priors," in *Proceedings of the Conference on Neural Nets Wirn10: Italian Workshop on Neural Nets*, Vietri sul Mare, Italy, February 2011.

[19] I. Sora, "Helping program comprehension of large software systems by identifying their most important classes," in *Proceedings of the International Conference on Evaluation of Novel Approaches to Software Engineering*, vol. 599, pp. 122–140, Barcelona, Spain, April 2015.

[20] I. Sora, G. Glodean, and M. Gligor, "Software architecture reconstruction: an approach based on combining graph clustering and partitioning," in *Proceedings of the 2010 International Joint Conference on Computational Cybernetics and Technical Informatics*, pp. 259–264, Timisoara, Romania, May 2010.

[21] D. S. Lekha and K. Balakrishnan, "Central attacks in complex networks: a revisit with new fallback strategy," *Physica A: Statal Mechanics and Its Applications*, vol. 549, pp. 1–19, 2020.

[22] M. Bellingeri, D. Bevacqua, F. Scotognella et al., "The heterogeneity in link weights may decrease the robustness of real-world complex weighted networks," *Scientific Reports*, vol. 9, Article ID 10692, 2019.

[23] Z. Qi, M. Z. Li, and Y. Deng, "A betweenness structure entropy of complex networks," pp. 1–18, 2014, http://arxiv.org/abs/1407.0097.

[24] M. Bellingeri, D. Bevacqua, F. Scotognella et al., "A comparative analysis of link removal strategies in real complex weighted networks," *Scientific Reports*, vol. 10, no. 1, Article ID 3911, 2020.

[25] Z. K. Zhang, T. Zhou, and Y. C. Zhang, "Tag-aware recommender systems: a state-of-the-art survey," *Journal of Computer Science and Technology*, vol. 26, no. 5, pp. 765–777, 2012.