Soft Comput (2018) 22:2583–2601
https://doi.org/10.1007/s00500-017-2513-y

METHODOLOGIES AND APPLICATION

A novel subgraph K^+-isomorphism method in social network based on graph similarity detection

Huan Rong[1] · **Tinghuai Ma**[2] · **Meili Tang**[3] · **Jie Cao**[4]

Published online: 15 February 2017

Abstract In this paper, we propose a novel K^+-isomorphism method to achieve K-anonymization state among subgraphs or detected communities in a given social network. Our proposed K^+-isomorphism method firstly partitions the subgraphs we have detected into some similar-subgraph clusters followed by graph modification conducted in every cluster. In this way, it is feasible to publish preserved structures of communities or subgraphs and every preserved structure actually represents a cluster of at least K subgraphs or communities which are isomorphic to each other. The contributions of this paper are listed as follows: On the one hand, we improve a maximum common subgraph detection algorithm, MPD_V, which is a core technique for graph similarity detection involved in partition phase of our proposed K^+-isomorphism method; on the other hand, with minor adjustment, we utilize some current techniques as an innovative combination to finish the partition and modification of similar-community cluster in K^+-isomorphism method. The experiments have shown that the improved MPD_V method has much better efficiency to search larger common subgraphs with acceptable performance compared with its prototype and other techniques. Moreover, our proposed K^+-isomorphism method can achieve the K-isomorphism state with less modification of original network structure, or lower anonymization cost compared to the current K-isomorphism method.

Keywords Social network · Subgraph similarity detection · Subgraph K-anonymization · Privacy preservation · Maximum common subgraph

Communicated by V. Loia.

✉ Tinghuai Ma
thma@nuist.edu.cn

Huan Rong
1227558210@qq.com

Meili Tang
meilitg@126.com

Jie Cao
cj@nuist.edu.cn

1 School of Computer, Nanjing University of Information Science and Technology, Nanjing 210044, Jiangsu, China

2 CICAEET, Jiangsu Engineering Centre of Network Monitoring, Nanjing University of Information Science and Technology, Nanjing 210044, Jiangsu, China

3 School of Public Administration, Nanjing University of Information Science and Technology, Nanjing 210044, China

4 School of Economics and Management, Nanjing University of Information Science and Technology, Nanjing 210044, China

1 Introduction

With the rapid development of mobile devices and Internet of things, social network has now obtained a huge amount of information, including the information flow between any two communicating entities forming a structure like "edge", as well as a diverse range of tuples consisting of labels and attributes to uniquely identify entities in network, or a "vertex", as a result, social network has now been equipped with relationships among users and personal data with significantly commercial value (Ma et al. 2015b; Bin and Sheng 2015b; Wen et al. 2015). Consequently, in order to prevent the private data mentioned above from misuse caused by malicious attacks or hackers, it is inevitable to pay more attention to privacy preservation (Praveena and Smys 2016).

Techniques of structural privacy preservation in social network can be classified into four types: Vertex anonymization (Doka et al. 2015; Chen et al. 2015; Bhagat et al. 2010)

divides several vertexes into a "super-node," each super-node containing at least K vertexes; subgraph anonymization (Cheng et al. 2010; Huda et al. 2013; Okada et al. 2014; Kiyomoto and Miyake 2014; Xiaoshuang and Masayuki 2015; Chakravorty et al. 2014; Bin et al. 2015a; Lv and Ma 2016; Wang et al. 2014; Zaghian and Bagheri 2016; Wu et al. 2010; Yang et al. 2014; Liu and Yang 2011; Zheng et al. 2015; Casas-Roma et al. 2014) means when using a particular sub-structure to re-identify specific vertexes and edges in original network, there always exist several ones whose structure are identical to each other; data perturbation (Ma et al. 2016a; Guo et al. 2014; Rajaei et al. 2015) randomly modifies social network so that attackers cannot accurately discover the initial structure of network; and finally, inference control (Zhang and Tang 2014) prevents evolution prediction of social network according to different prediction models.

Among all the techniques mentioned above, what we focus on most in this paper is subgraph anonymization. Bin et al. (2015a), Lv and Ma (2016), Wang et al. (2014) and Ma et al. (2015a) propose a K-degree anonymization method to resist malicious attacks which use vertex degree as background knowledge; Zaghian and Bagheri (2016) present a method for neighbor-graph anonymization, so that for 1-neighbor-graph of any vertex, there exist at least $K-1$ vertexes whose 1-neighbor-graph have the same composition; the K-symmetry subgraph anonymization method proposed by Wu et al. (2010) has been presented to make the initial social network become a totally symmetric one; Yang et al. (2014) have proposed K-automorphism concept for social network privacy preservation; the privacy leakage caused by "weight-package" has been studied to prevent disclosure (Liu and Yang 2011; Zheng et al. 2015; Casas-Roma et al. 2014).

K-isomorphism is a famous subgraph anonymization technique proposed in Cheng et al. (2010) whose general process can be concluded as shown in Fig. 1. As shown in Fig. 1, in partition phase, the K-isomorphism algorithm splits the whole network into K non-overlapped subgraphs $\boldsymbol{N} = \{\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3, \ldots \boldsymbol{g}_k\}$, then in modification phase, the algorithm modifies these K subgraphs to achieve an isomorphic state by addition and deletion of vertexes and edges. Finally, it outputs a preserved network consisting of K-subgraphs isomorphic to each other which means for any sub-structure, there exist at least $K-1$ consistent ones, ensuring that the possibility of re-identification of any particular vertex or subgraph is less than $1/K$.

A large amount of works about the optimization of K-isomorphism have been conducted; in Huda et al. (2013), a novel data publishing scheme has been proposed to maintain the high utility of initial social network; a strategy of suppressing the addition of noisy edges which expands the distance of neighbored pairs of nodes in original network has been proposed Okada et al. (2014) and Kiyomoto and Miyake (2014) have presented a method to select the best value of K for anonymization in different size of networks; Xiaoshuang and Masayuki (2015) have used the principle of clustering to preserve a social network in order to achieve the final K-isomorphism state; last but not the least, Chakravorty et al. (2014) have comprehensively analyzed a paralleled way to improve the K-isomorphism algorithm so that it can process the social network with huge data cardinality.

2 Problem definition

All the techniques about subgraph K-isomorphism mentioned in Introduction section have some common features: Firstly, the object of privacy preservation is the whole social network, which means in modification phase, all the vertexes will be considered for addition or deletion, in turn, what will be available for users after data publishing is a whole preserved network structure without any unique label or attribute of vertex. Secondly, at the initial step, algorithms in Cheng et al. (2010), Huda et al. (2013), Okada et al. (2014), Kiyomoto and Miyake (2014), Xiaoshuang and Masayuki (2015) and Chakravorty et al. (2014) need to partition the whole social network into several subgraphs as original datasets for modification which means the modification is based on network partition. In short, the concept of K-isomorphism, or subgraph anonymization, in current works is all targeted at social network itself.

Unfortunately, consider the following scenario:

Problem definition We have proposed the work of community detection in C-DBLP by BSCHEF framework in Ma et al. (2016b). When finishing the community detection process, if we still hope to achieve subgraph anonymization, more explicitly, the K-isomorphism of communities, then the object of privacy preservation should only be focused on vertexes and edges covered by community structure not the whole network. More seriously, as the communities have been detected already, the random partition of the whole network into several subgraphs as original datasets for anonymization is not applied any more. Moreover, after data publishing, what would be really available for users are several different structures of communities but not the whole network, and every outputted community structure actually represents a group of communities. Consequently, if we still utilize the current K-isomorphism method demonstrated in Fig. 1 to preserve a group of communities or subgraphs, the current works mentioned above are not feasible.

In this paper, we propose a novel K^+-isomorphism method particularly targeted at the anonymization of community or subgraph which means for $\boldsymbol{m}$ detected communities, we partition them into $\boldsymbol{n}$ similar-subgraph clusters and in every cluster, there exist at least $\boldsymbol{K}$ communities isomorphic to each other after modification. In this way, we are able to achieve the anonymization state among commu-

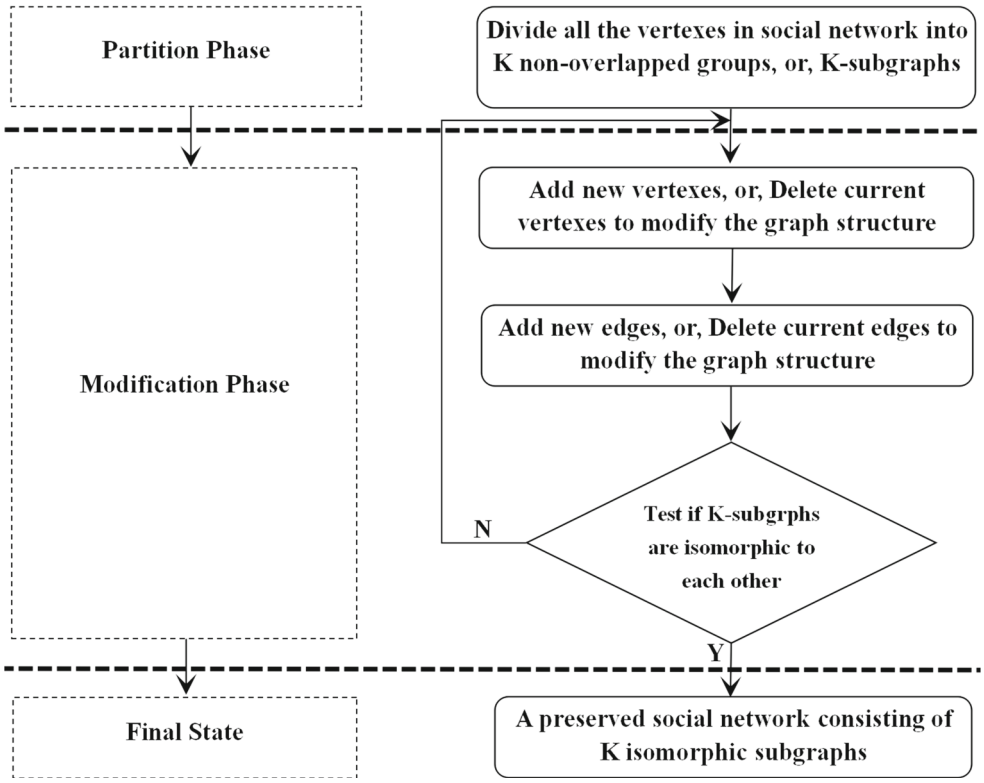

Fig. 1 General steps about K-isomorphism

nities, or subgraphs (in the following part of paper, the concept of "subgraph" and "community" can be switched equally).

Generally, we make a reference to the process presented in Fig. 1, utilizing some related techniques to modify the whole steps. Figure 2 demonstrates the partition phase of K^+-isomorphism method. Firstly, it detects communities by BSCHEF (Ma et al. 2016b) to obtain initial community set $\boldsymbol{C}$ consisting of $\boldsymbol{m}$ detected communities, followed by the partition of whole set $\boldsymbol{C}$ into $\boldsymbol{n}$ clusters based on graph similarity detection and other related algorithms. Every cluster contains at least $\boldsymbol{K}$ communities. Then, Fig. 3 illustrates the process of modification phase in K^+-isomorphism method, different from that of Fig. 1, the K^+-isomorphism method passes all the K-community clusters to modification phase, respectively. Through the addition and deletion of vertexes and edges, we ensure that all the communities located in the same cluster are isomorphic to each other.

The contributions or innovation of this paper are listed as follows:

1. Improve the maximum common subgraph detection algorithm, MPD_V, which is a core technique for graph similarity detection involved in partition phase of our proposed K^+-isomorphism method.
2. With our minor modification, use some current algorithms as an innovative combination to finish the process of similar-community cluster partition and modification in K^+-isomorphism method.

For the rest of this paper, in Sect. 3, we will discuss in detail about the improved algorithm MPD_V for maximum common subgraph detection; in Sect. 4, we will explain the specific process of partition phase shown in Fig. 2 as well as that of modification phase shown in Fig. 3 including details about the critical algorithms we utilized. Last but not the least, in Sect. 5, we firstly test the efficiency and effectiveness of MPD_V compared with its prototype as well as other representative maximum common subgraph detection algorithms. Finally, we compare the anonymization cost of our proposed K^+-isomorphism method with other K-isomorphism techniques when achieving the state of K-anonymization, more specifically K-isomorphism, in their own meaning.

3 Subgraph similarity detection

Nowadays, in order to evaluate the similarity between two given graphs, more and more researchers have turned to the application of maximum common subgraph whose goal is to search the largest common part between query graph and data graph. As shown in Fig. 4, with the given query graph and data graph, Fig. 4c shows the maximum common subgraph between Fig. 4a, b.

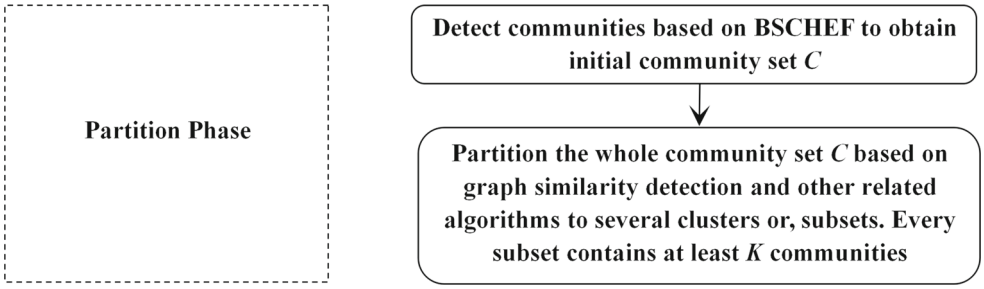

Fig. 2 Partition phase of K^+-isomorphism method

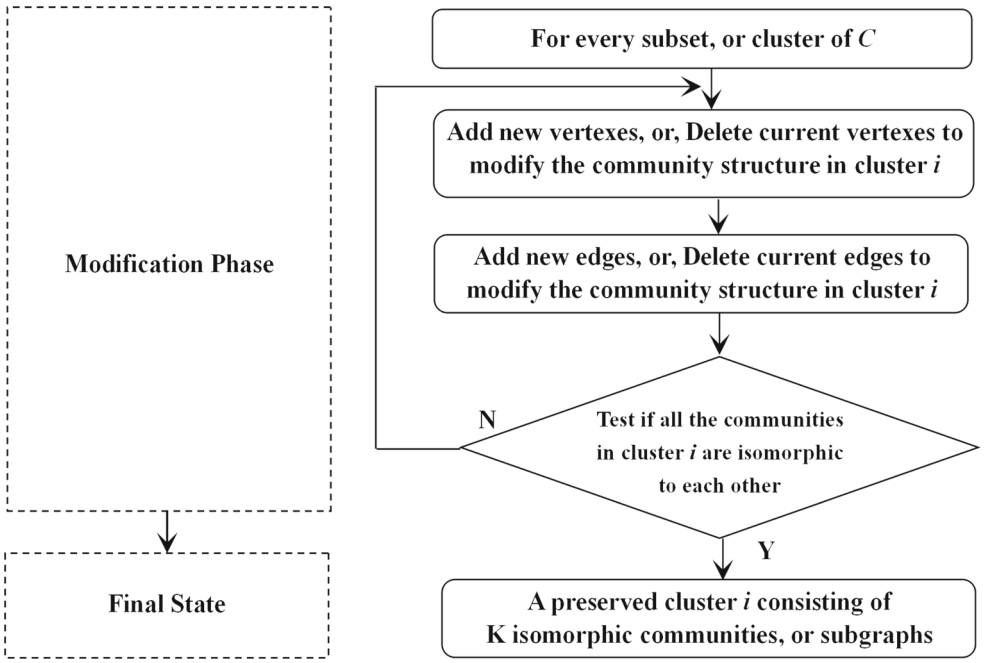

Fig. 3 Modification phase and final state of K^+-isomorphism method

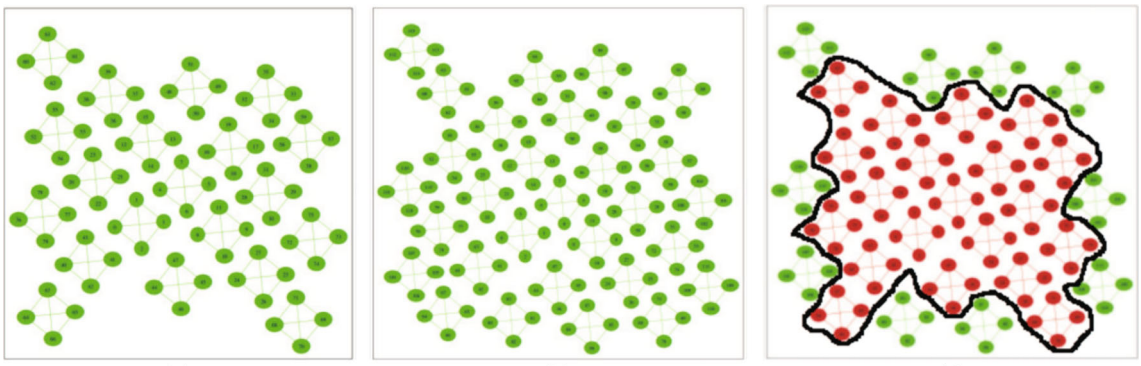

Fig. 4 An example of similarity evaluation by maximum common subgraph. **a** Query graph. **b** Data graph. **c** Maximum common subgraph

Then, we can calculate the graph similarity between Fig. 4a and Fig. 4b by Eqs. (1) and (2) according to the size of searched common subgraph, so that it is feasible to get a more accurate result for similarity in form of "number." The definition of maximum common subgraph is listed below:

Definition 1 (*maximum common subgraph*) With query graph Q and data graph G, g is a maximum common subgraph between Q and G if and only if g is subgraph isomorphic to Q and G. In addition, for any other g' which

is subgraph isomorphic to Q and G, $|\ g\ | > |\ g'\ |$, where the symbol "$|\cdot|$" represents the number of vertex.

Mallek et al. (2015) have used maximum common subgraph to detect graph similarity when detecting communities in social network; Hu et al. (2013) have explored similar biological molecular composition by maximum common subgraph; In Raymond and Willett (2002), a revision of maximum common subgraph for chemical structures matching has been conducted.

Equation 1 (Graph similarity calculation I) *Suppose max* $(\|Q\|, \|G\|)$ *is the maximum number of vertex in query graph or data graph,* $\|\ mcs(Q,G)\ \|$ *means the number of vertexes in searched common subgraph, sim(Q,G)* $\in$ *[0,1].*

$$\textbf{sim(Q,G)} = \frac{\|mcs(Q,G)\|}{max(\|Q\|, \|G\|)} \tag{1}$$

Equation 2 (Graph similarity calculation II) *Suppose Vmax is the largest common vertex set between query graph Q and data graph G; Emax is the largest common edge set between query graph Q and data graph G;* V_Q, V_G *are two vertex sets of Q, G, respectively, then, let Gmax(Vmax , Emax)=mcs,* $0 < \varphi < 1$, *then* $\alpha = \|Vmax\|/\|V_Q\|$ *describes to what extent G is a subset (subgraph) of Q, and* $\beta = \|Vmax\|/\|V_G\|$ *describes to what extent Q is a subset (subgraph) of G.*

$$\textbf{sim(Q, G)} = \varphi \times \textbf{min}(\alpha, \beta) + (\textbf{1} - \varphi) \times \textbf{max}(\alpha, \beta) \tag{2}$$

In this section, we mainly demonstrate a novel maximum common subgraph detection algorithm MPD (Wang and Maple 2005) whose principle has been illustrated in Sect. 3.1 as well as our improved version MPD_V which optimizes the search result based on connectivity to enhance the efficiency of MPD.

3.1 MPD algorithm prototype and its dilemma

Given query graph and data graph shown in Fig. 5, numbers in bracket represent the degree of a node and we temporarily use degree as the label of each node. In Gouda and Hassaan (2012), it has pointed out that according to the *refinement condition* of *Ullmann* algorithm (Ullmann 1976), a vertex $\boldsymbol{u}$ in query graph can be mapped to vertex $\boldsymbol{v}$ in data graph if and only if they have the same label and the degree of $\boldsymbol{u}$ is no more than that of $\boldsymbol{v}$, which means in Fig. 5, a node with degree $\boldsymbol{d}$ in data graph can be mapped to that in query graph whose degree is no more than $\boldsymbol{d}$. Under such vertex mapping strategy, MPD algorithm's prototype firstly makes the "match pair" data structure shown in Table 1 as a preprocess phase before the start of maximum common subgraph search. A match pair like (a,1) means the vertex "a" in data graph can be mapped to vertex "1" in query graph.

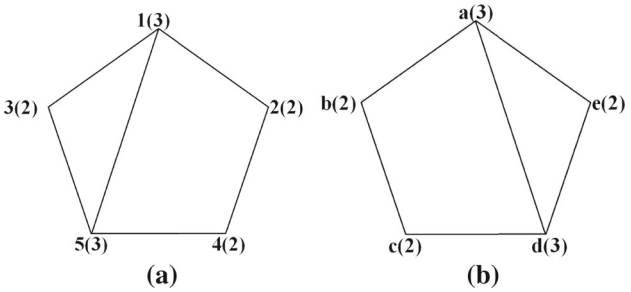

Fig. 5 Two given graphs. **a** Query graph. **b** Data graph

Table 1 Match pair data structure of query graph and data graph in Fig. 4

Data graph node	Match pairs
a	$(a, 1), (a, 2), (a, 3), (a, 4), (a, 5)$
b	$(b, 2), (b, 3), (b, 4)$
c	$(c, 2), (c, 3), (c, 4)$
d	$(d, 1), (d, 2), (d, 3), (d, 4), (d, 5)$
e	$(e, 2), (e, 3), (e, 4)$

Next, according to the match pairs shown in Table 1, for particular data graph nodes such as a, b, c, d, e, MPD splits all of their match pairs, picks up query graph nodes and stores them in a linked list with the addition of a "NULL" node at the tail. Each linked list is marked as a "trie_link" for one row. As a result, with given match pairs in Table 1 we can construct a "layered-linked" search space for a given pair of query graph and data graph illustrated in Fig. 6 as an example. In Fig. 6, the solid arrow lines represent the "line selection in depth" and the dotted arrow lines represent the "breadth-substitution" in back-tracking. Such two processes will be demonstrated below.

In such search space, MPD firstly selects a data node and a query node in one row as an initial match pair (like $(a, 1)$ in Fig. 6), then it selects another data graph node in a new row *only* in downward direction if and only if two data graph nodes in different rows have an edge in data graph shown as the solid arrow line at left side (for example, from a to d).

Moreover, every time when MPD selects a new data graph node in a new row in downward direction, it will also select a related new query graph node with the same constraint mentioned above to expand query graph edge like the solid arrow line from 1 to 5 at the right side. In this way, MPD expands the searched path in "depth," or downward direction firstly.

When the expansion in depth cannot be done any more, the back-tracking process will be triggered, shown as the dotted arrow line in Fig. 6, which firstly explores a new query graph node in the last trie_link row and replaces the old query graph node with a new one at the tail of current searched match

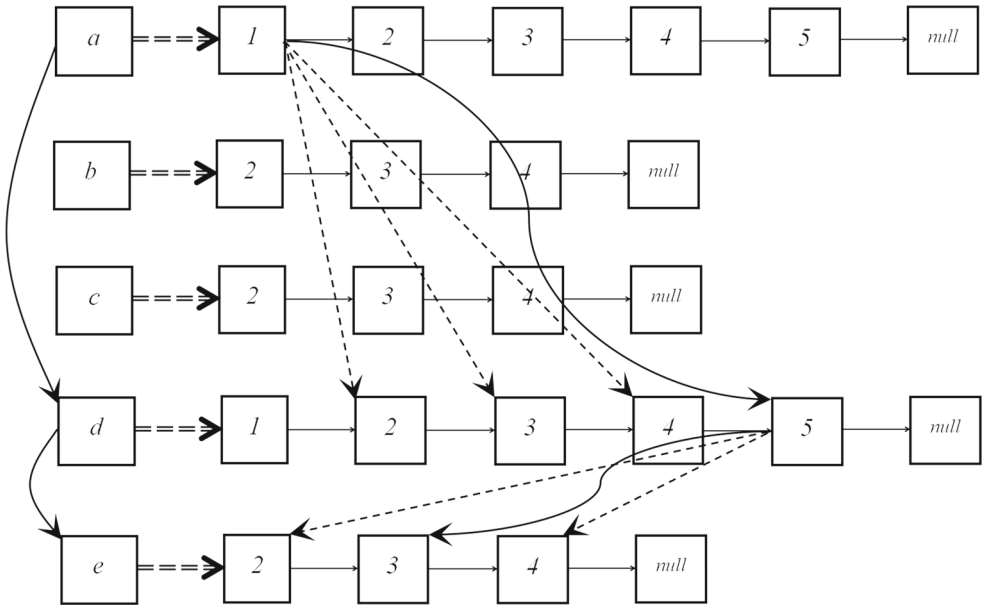

Fig. 6 Layered-linked search space built from Table 1 by MPD

pair path, followed by a new continuous deep expansion; otherwise, the algorithm goes back in depth and selects other query graph node in above trie_link rows. In this way, MPD expands searched path in breadth by selecting a new query graph node in current row when doing back-tracking.

In conclusion, with the help of line selection in "depth" and breadth-substitution in back-tracking, MPD can search all valid mapping-paths between query graph and data graph following such "Depth-First, Breadth-Second" rule.

Finally, MPD will select the longest searched common subgraphs and calculate the similarity between query graph and data graph by Eqs. (1) or (2). It is evident to be found that the larger the value calculated by Eqs. (1) or (2), the higher the similarity between query graph and data graph which means there exist more matched vertexes between two graphs.

Unfortunately, the "Depth-First, Breadth-Second" search process of MPD illustrated above has a big problem. For example, Fig. 7 shows the layered-linked search space built from the query graph and data graph shown in Fig. 5. After using $(a, 1)$ as an initial match pair, among the entire result space, it can be found that MPD has contained (i) $(a, 1) - (b, 2) - (c, 4) - (d - 5) - (e, 3)$, (ii) $(a, 1) - (d, 5) - (e, 3)$, (iii) $(a, 1) - (e, 3)$ three different paths separately and path(i) is one of the final searched maximum common traces. The similarity will be calculated on path(i) by Eqs. (1) or (2).

However, the query graph and data graph shown in Fig. 5 have covered all above three paths, which means the MPD outputs have damaged connectivity because of the constraint of the height of the "layered-linked" search space; in turn, such drawback will decrease the efficiency of graph similarity calculated by Eqs. (1) or (2). As a result, an improvement of MPD (MDP Verification, MDP_V) needs to be done to verify MPD's result set, or, merging the common traces into a new connected one after each search round triggered by an initial match pair but still within the size of both query graph and data graph, so that, we will finally obtain a ***maximum-connected*** common subgraph.

Taking Fig. 5 as an example, MPD_V will finally output the whole data graph and query graph as the searched maximum-connected common subgraph pair with the addition of result verification.

3.2 Improved version MPD_V

The improved version, MPD_V, has added a new result set verification phase after each search round triggered by a particular initial match pair, while the whole search process still remains the same as that of MPD. All the different maximum common subgraphs outputted by search process are treated as seeds, meanwhile the rest common subgraphs will be split up into the form of non-repeated [data graph edge, query graph edge] edge mapping pairs, taking the role of original candidate edge mapping set used for seed expansion based on connectivity. The general step of result set verification in MPD_V is shown in Algorithm 1.

In line 1, the Algorithm 1 firstly obtains the number of vertex in seed as *seed_size* and its initial match pair in line 2, then in line 3, the algorithm *buildMaterial*(*seed_size*, *start_match_pair*) uses all the non-maximum common subgraphs which have the same initial match pair as *start_match_pair* to construct the original candidate edge mapping

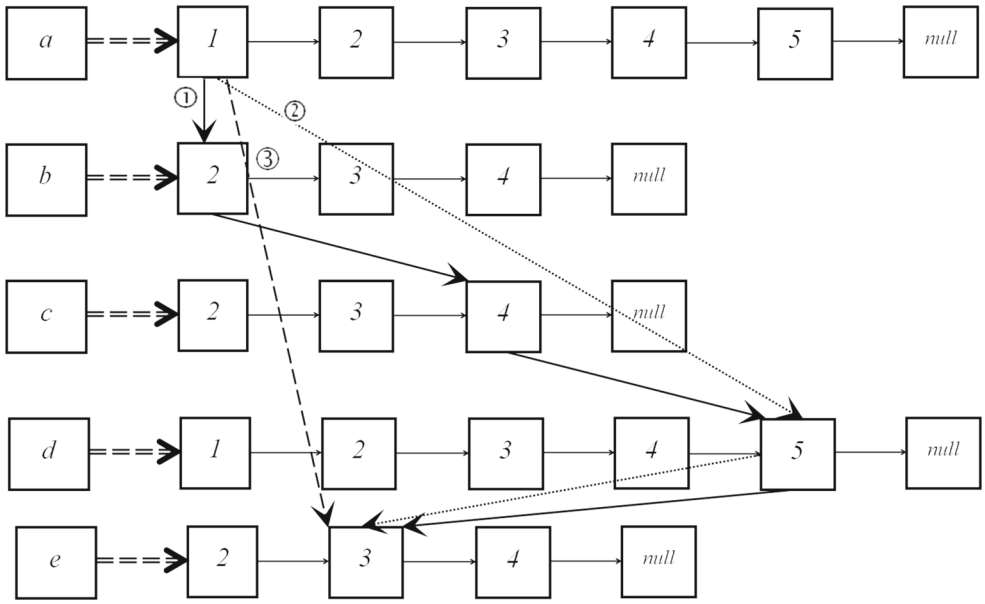

Fig. 7 Searched path in layered-linked search space of Fig. 5

Algorithm 1 Result Set Verification of MPD_V

Require: *seed* after a search round.
Query_graph_edges_size.
Ensure: Verify the searched result set for a given seed triggered by an initial match pair.
1: Make *seed_size* as the number of vertex in *seed*
2: Get *start_match_pair* of seed
3: buildMaterial(*seed_size*, *start_match_pair*)
4: growByMaterial(*seed*, *Query_graph_edges_size*)

Algorithm 2 Procedure **buildMaterial** of building original candidate edge mapping set

Require: *seed_size*: the number of vertex in *seed*.
start_match_pair:the initial match pair triggering current search round.
(Note: this algorithm is the third step mentioned in Algorithm 1)
Ensure: The building of original candidate edge mapping set.
Select all common subgraphs with size ≥ 2
2: AND size $<$ *seed_size*
AND has *start_match_pair*
4: from database as *cs* set
while still has more common_subgraph in cs **do**
6: Get *new_common_subgraph* from *cs*
Make *edges* from *new_common_subgraph* as tuple like:
8: [*fromDataGraphNode, toDataGraphNode*,
fromQueryGraphNode, toQueryGraphNode]
10: **if** current tuple is already exist in *edges* **then**
Delete current tuple from *edges*
12: **end if**
store *edges* into database as original candidate edge mapping set;
14: **end while**

set. Finally, the expansion of seed by candidate edge mapping set will be conducted in *growByMaterial*(*seed*, *Query_graph_edges_size*) shown as line 4.

The process of *buildMaterial()* for building original candidate edge mapping set is shown in Algorithm 2, and the detail of *growByMaterial()* for seed expansion is presented in Algorithm 3.

As shown in Algorithm 2, lines 1–4 collect all the common subgraphs whose number of vertex, or *size*, is smaller than that of the given *seed*. In line 5, the algorithm traverses all the collected common subgraphs and splits them into non-repeated candidate mapping-edge-pairs like [*fromDataGraphNode, toDataGraphNode, fromQueryGraphNode, toQueryGraphNode*] in lines 6–9 before storing the final mapping pairs into set "*edges*" in line 10–14. The expansion of seed, or the specific steps of *growByMaterial()*, is shown in Algorithm 3.

In Algorithm 3, line 9 checks whether the given seed contains data graph edge or query graph edge obtained in line 7 and line 8, if so, the algorithm finds new mapping-edge-pair in line 10, otherwise, it checks whether both the obtained data graph edge and query graph edge are all connected to seed in line 12. Line 13 adds the valid mapping-edge-pair to seed if the algorithm passes the check in line 12.

Every time when the addition of edges selected by line 12 happens in line 13, the algorithm will immediately check whether the number of vertex of seed's updated state has exceeded that of query graph in lines 14–17, if the check passes, two new added edges will be removed from seed in line 15 and the algorithm moves on to select a new candidate mapping-edge-pair in line 16, otherwise, the "hasAdded" field will be changed to true value in line 18 representing that the seed has been modified in current expansion round.

Algorithm 3 Procedure **growByMaterial** of seed expansion

Require: *seed*: the searched maximum common subgraph.
query_graph_edge_size:the number of edges in the query graph.
(Note: this algorithm is the fourth step mentioned in Algorithm 1)
Ensure: The input *seed* has been expanded by original candidate edge mapping set.
while true **do**
Make *hasAdded* = false
3: Make *candidate_edges* from database;
while still has more *edge* in *candidate_edges* **do**
Get a new *candidate_edge* as
6: [*fromDataGraphNode, toDataGraphNode, fromQueryGraphNode, toQueryGraphNode*]
Make *data_graph_edge* from *candidate_edge*
Make *query_graph_edge* from *candidate_edge*
9: **if** *seed* contains *data_graph_edge* or *query_graph_edge* **then**
continue
end if
12: **if** *data_graph_edge* AND *query_graph_edge* all connected with *seed* **then**
Add [*data_graph_edge→query_graph_edge*] mapping to seed
if seed has more vertex than query_graph **then**
15: Delete new added *mapping* from seed
continue
end if
18: Make *hasAdded* = true
Delete the *candidate_edge* from database
end if
21: **if** length of *seed* equal to *query_graph_edge_size* **then**
store *seed* as a new verified one into database
Delete all new added *edges* from *seed*
24: break
end if
end while
27: **end while**
if *hasAdded* equal to true **then**
store *seed* if has not stored a same one
30: Delete all new added *edges* from *seed*
continue
else
33: break
end if

In lines 21–25, the algorithm checks if the edge size of current seed is equal to that of query graph, if so, the query graph is (sub) graph isomorphic to data graph and the current expansion round can be stopped.

In lines 28–34, according to the value of "hasAdded," the algorithm checks if it is necessary to continuously use the remaining candidate edge mapping set for another round of seed expansion in order to prevent the loss of other valid maximum-connected common subgraphs.

It is worthwhile to mention that as a particular mapping-edge-pair can be used only once, for a given seed, the new added edges will be removed from the original candidate edge mapping set in line 19. As a result, for different seeds even generated by a same initial match pair, the building of the whole original candidate edge mapping set in Algorithm 2 is always needed to be done again before the start of another seed expansion shown in Algorithm 3.

4 K^+-isomorphism method for community detection scheme

In this section, we mainly discuss details about the partition phase and modification phase, respectively, of our proposed K^+-isomorphism method shown in Figs. 2 and 3. Section 4.1 illustrates the principle of how we partition all the subgraphs, or detected communities into several similar-subgraph or similar-community clusters as well as the detail of ***MCA*** (matrix clustering algorithm) which we have applied in partition phase. Section 4.2 demonstrates the detail about the modification phase of our proposed K^+-isomorphism method.

4.1 Partition for similar-community clusters

4.1.1 Bi-clustering and sub-matrix partition

In order to complete the whole partition phase in K^+-isomorphism method to obtain several similar-community clusters, we firstly utilize the principle of "bi-clustering" in Zhang et al. (2010) to build our partition model. In bi-clustering, as shown in Fig. 8, all the elements in the same cluster are the most similar to each other, or, close in "distance"; however, what is different from traditional clustering is that the condition for the forming of clusters, or the standard to judge which elements are the most similar to each other, is not applied globally. In other words, for ***Cluster A*** in Fig. 8, elements 1, 2, 3 are similar to each other if and only if "Condition I" has been satisfied, otherwise, ***Cluster A*** will not be maintained any more when "Condition I" has been changed or removed. Therefore, in a more serious way, bi-clustering utilizes different attribute sets or types of similarity to build every cluster; however, under a given "Condition," all the elements located in the same cluster are the most similar or closest to each other.

Analogously, back to the partition phase in K^+-isomorphism method, the principle of forming of clusters in "bi-clustering" shown in Fig. 8 is also applied to our partition problem model. In Fig. 8, if we consider every element in a cluster as a kind of subgraph, or, community outputted by a given community detection scheme, then the meaning of ***Cluster B*** is that, when given "Condition II," community 4, 5, 6, 7 are the most similar to each other which means under a given target community subset $\boldsymbol{T}$, the graph similarity between community 4, 5, 6, 7 and $\boldsymbol{T}$ is the highest. However, when the subset $\boldsymbol{T}$ (Condition II) has been changed, the ***Cluster B*** will not probably be maintained any more because it

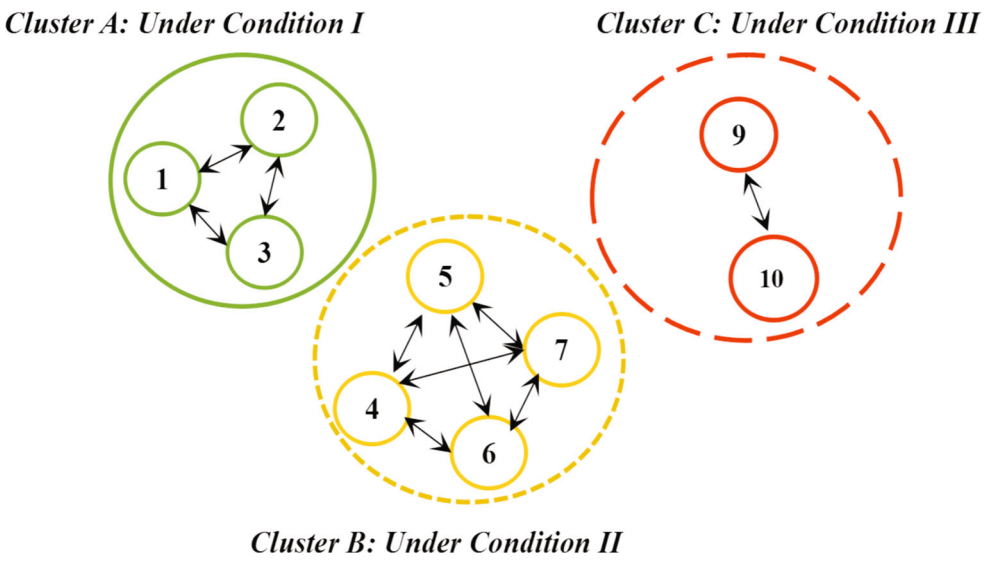

Fig. 8 Principle of "bi-clustering"

cannot be ensured the graph similarity between community 4, 5, 6, 7 and any subset $\boldsymbol{T}$ is always the highest. In short, a group of communities can only be considered as similar enough to be located in the same cluster if and only if a particular target community subset $\boldsymbol{T}$ has been given, or the so-called Condition.

Moreover, Madeira and Oliveira (2004) have proposed a "two-dimensional" matrix model for converting the "bi-clustering" mentioned above into sub-matrix partition problem based on the same principle. As shown in Fig. 9, given a $m \times n$ matrix $\mathrm{A} = [\mathrm{a}_{i,j}]_{m\times n}$, in the row direction, it represents $\boldsymbol{m}$ entities such as $\boldsymbol{m}$ source subgraphs, or communities. In the column direction, it represents $\boldsymbol{n}$ attributes of every entity in a row, say $\boldsymbol{n}$ target subgraphs for a given source subgraph in row direction. Then, for any element $\mathrm{a}_{i,j}$ in matrix $\mathrm{A} = [\mathrm{a}_{i,j}]_{m\times n}$, it means the exact value of attribute $\boldsymbol{j}$ that entity $\boldsymbol{i}$ owns, while in current context the element $\mathrm{a}_{i,j}$ represents the graph similarity between source subgraph $\boldsymbol{i}$ and the target subgraph $\boldsymbol{j}$.

A sub-matrix partitioned in Fig. 9 means that, for some source communities in row direction, they are the most similar to each other when giving a particular target community subset $\boldsymbol{T}$ in column direction. The subset $\boldsymbol{T}$ in column direction is just the "Condition" that supports the forming of a cluster consisting of source communities in row direction. Therefore, ***Sub-matrix A-C*** in Fig. 9 is equal to ***Cluster A-C*** in Fig. 8 based on the same principle. A sub-matrix is just the similar-community cluster that we finally want to obtain in partition phase of K^+-isomorphism method.

Seriously, assume set $\boldsymbol{C} = \{1, 2, \ldots, j\}$ and $\boldsymbol{R} = \{1, 2, \ldots, i\}$ are the index set in column and row direction of matrix $\mathrm{A} = [\mathrm{a}_{i,j}]_{m\times n}$, respectively, then, the definition of bi-clustering based on sub-matrix partitioned from source two-dimensional matrix $\boldsymbol{A}$ is explained as follows:

Definition 2 (*Bi-clustering based on sub-matrix partition*) Given matrix $\mathbf{A}_{i\times j}$, select $C_1, C_2, C_3, \ldots C_j \subset C$ and $R_1, R_2, R_3, \ldots R_i \subset R$, let $C_t \in \{C_1, C_2, C_3, \ldots C_j\}$ and $R_S \in \{R_1, R_2, R_3, \ldots R_i\}$, then, $[\mathrm{a}_{ij} : i \in R_s, j \in C_t]$ made up of row vector $\{a_i \mid a_{ij} \in a_i, i \in R_s, j \in C\}$ and column vector $\{\beta_j \mid a_{ij} \in \beta_j, i \in R, j \in C_t\}$ is a sub-matrix marked as $A[R, C]$ detected from A. The goal of bi-clustering based on sub-matrix partition is to find a sub-matrix $A[R, C]$ induced from $R_s \subset R, C_t \subset C$ whose elements have the highest similarity. $A[R, C]$ is also called as a bi-directed cluster.

Generally, for a given two-dimensional matrix $\boldsymbol{A}$, the sub-matrix partition scheme can be concluded as (a)–(f) shown in Fig. 10. Due to the feature of BSCHEF (Ma et al. 2016b) whose community detection scheme dose not contain any overlapped community as well as the decomposition process of ***MCA*** algorithm which ensures all the rows and columns of original matrix can be used only once, we select Fig. 10b row–column exclusive model in this paper. The detail of matrix clustering algorithm (***MCA***) is discussed in next section.

*4.1.2 The application of matrix clustering algorithm(**MCA**) in cluster partition phase*

Unfortunately, the bi-clustering model based on sub-matrix partition proposed in Madeira and Oliveira (2004) is only applied for "binary-value" matrix consisting of 0 and 1 value; as a result, we utilize "Z-score" method to normalize every row of the initial matrix $\boldsymbol{A}$ converting it into a "binary-value" one. Then, the whole matrix will be like what has been presented in Fig. 11. In order to partition sub-matrix efficiently, we utilize ***MCA*** algorithm proposed in Jiang et al. (2010), a novel "matrix clustering" method particularly for "binary-

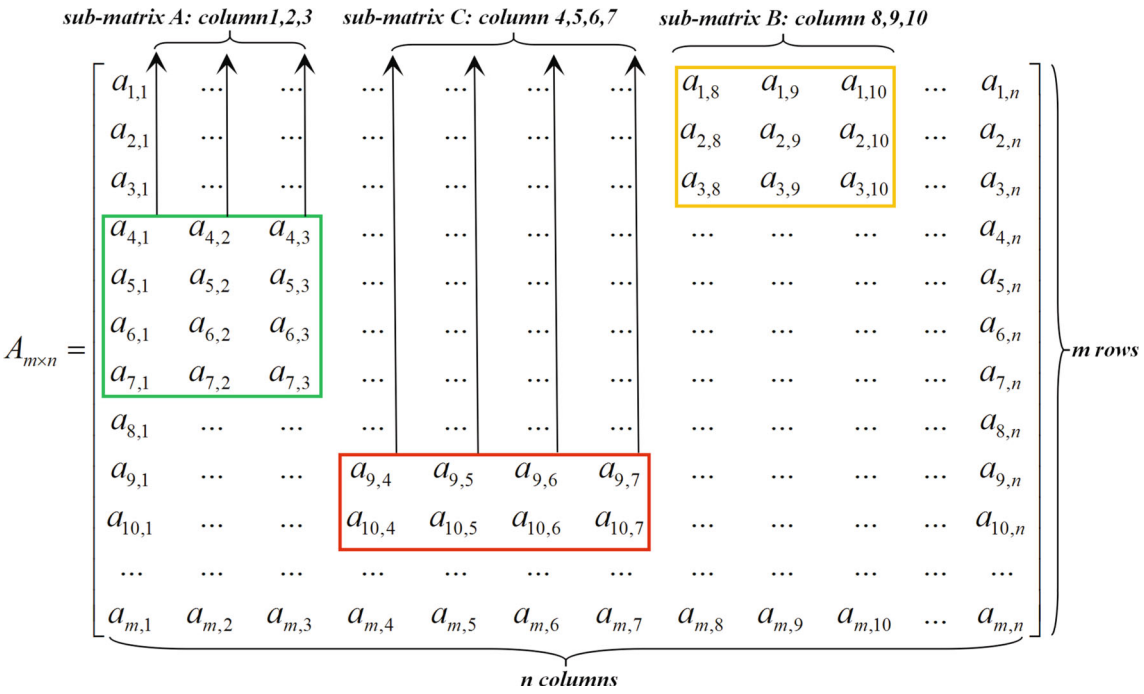

Fig. 9 Sub-matrix related to clusters in Fig. 8

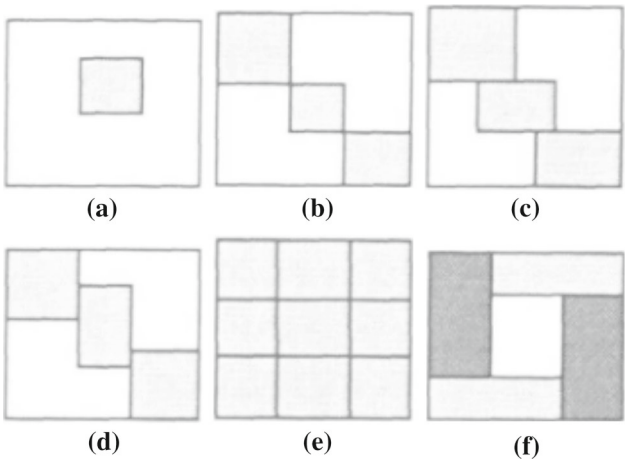

Fig. 10 Conclusion of sub-matrix partition scheme. **a** Single model. **b** Row–column exclusive model. **c** Row exclusive model. **d** Column exclusive model. **e** Chessboard model. **f** Fence model

value" matrix based on maxing the density of sub-matrix. ***MCA*** has been widely used in recommendation system of social network (Kuo et al. 2013). The definition of matrix density is listed as follows:

Definition 3 (*Matrix density*) Suppose the sub-matrix $A_{m1,n1}$ is partitioned from matrix $A_{m\times n}$, mark the number of value 1 in $A_{m1,n1}$ as $\sum M_{m1,n1}$, then, the matrix density of sub-matrix $A_{m1,n1}$ is $\Psi = \sum M_{m1,n1}/m1 * n1$, where $\Psi \in [0,1]$.

Figure 11 presents two main steps of shrink or partitioning and decomposition in one round of ***MCA*** algorithm.

In Fig. 11a, ***state 1*** of the shrink process in ***MCA*** represents any state before shrink or partitioning of matrix and ***state 2*** represents any sub-matrix after partitioned. The process of shrink from ***state 1*** to ***state 2*** in ***MCA*** demonstrated in Fig. 11a is based on the expansion and cut of rows and columns from the initial matrix which can be concluded as "Row expansion, row cut; Column expansion, column cut." The goal of such shrink process in ***MCA*** is to maximize the density (see ***Definition III***) or the number of 1 value of a partitioned sub-matrix during current partitioning round.

In addition, one round of shrink process from ***state 1*** to ***state 2*** will be stopped when the density of sub-matrix in ***state 2*** reaches a maximum value, or cannot be increased any more. Then we will obtain a final sub-matrix which represents a similar-community cluster containing communities in row direction with the ***support condition*** of target community subset ***T*** in column direction.

Every time when a final sub-matrix is partitioned, the rows and columns involved in ***state 2*** will be removed from the initial matrix shown in Fig. 11b, in this way, we can ensure that all the rows and columns will be used only once which means the sub-matrix detection pattern of ***MCA*** follows the row–column exclusive model presented in Fig. 10b. Then, another round of shrink or partitioning process in Fig. 11a will be conducted again based on the remaining decomposed matrix.

The detailed steps of ***MCA*** algorithm will be discussed in Algorithm 4 before which we firstly illustrate the explicit steps of the whole partition phase to obtain similar-

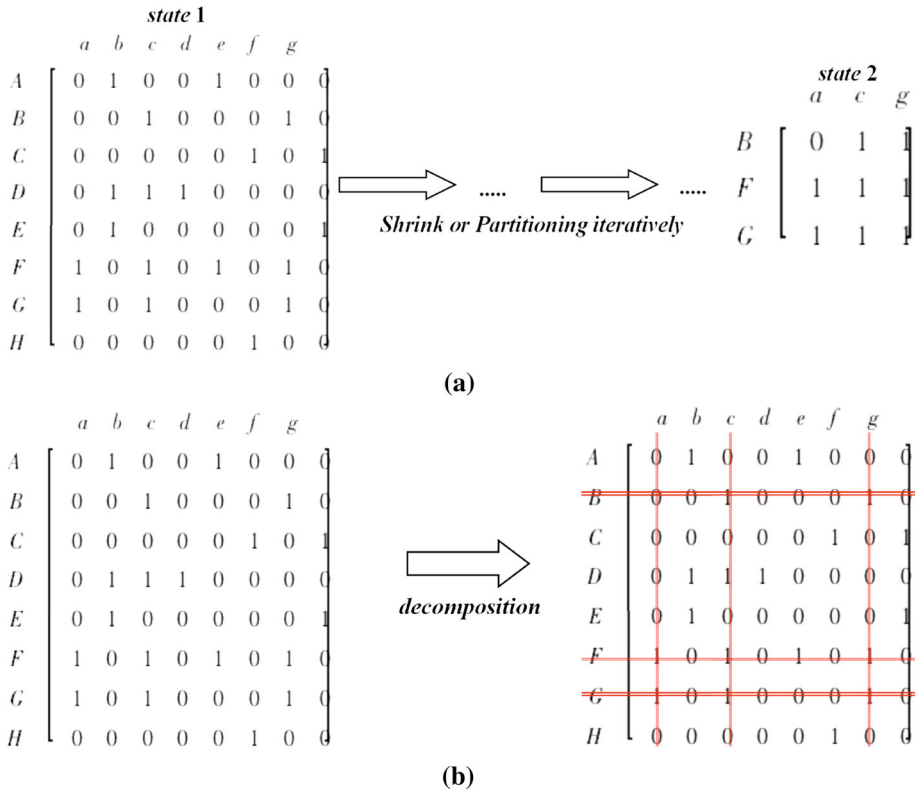

Fig. 11 Two main steps in one round of ***MCA*** algorithm. **a** The shrink or partitioning process of ***MCA*** algorithm. **b** The decomposition process of ***MCA*** algorithm

community clusters in our proposed K^+-isomorphism method shown in Fig. 12, a more specific version of Fig. 2.

For K^+-isomorphism method, as the communities have been already detected, it directly takes these subgraphs as individual elements, in this way, the random partition of the whole vertex set of original network $\boldsymbol{G} = \langle \boldsymbol{V}, \boldsymbol{E} \rangle$ mentioned in K-isomorphism method will not be done.

Then, K^+-isomorphism method calculates the similarity of every pair of communities with the help of MPD_V to make a similarity matrix $\boldsymbol{S}$ whose element $\boldsymbol{S}_{ij}$ represents the similarity between *community_i* and *community_j*. At this stage, as shown in Fig. 9, what should be done next is to detect sub-matrix by the process of shrinking and decomposition of ***MCA*** which have been demonstrated in Fig. 11. The decomposition is conducted on the similarity matrix every time when a final sub-matrix is detected.

Every final partitioned sub-matrix is just a similar-community cluster whose physical meaning is that for all the subgraphs in the row direction, they are similar to each other under the condition of target communities in column direction.

What should be noticed is that, due to the feature of ***MCA***, the original similarity matrix $\boldsymbol{S}$ is needed to be converted to a binary-value one at the beginning of ***MCA*** consisting only of 1 or 0 so that ***MCA*** can detect sub-matrix by density, or the number of 1 located in a sub-matrix. The binary-value matrix of $\boldsymbol{S}$ is marked as $\boldsymbol{S}$**'**, and we use Z-score method to normalize every row of $\boldsymbol{S}$ to obtain 0–1 values.

In Fig. 12, firstly we utilize BSCHEF (Ma et al. 2016b) to detect communities in a given social network $\boldsymbol{G} =< \boldsymbol{V}, \boldsymbol{E} >$. Then we use MPD_V, an improved algorithm (see **Algorithm 1**), to detect the maximum-connected common subgraph between any two communities discussed in Sect. 3 in order to calculate the graph similarity according to Eqs. (1) or (2) by which we build a similarity matrix $\boldsymbol{S}$ in the third step whose element $\boldsymbol{S}_{ij}$ represents the graph similarity between *community_i* and *community_j*.

It is important to notice that we put all the communities themselves in the column direction of similarity matrix $\boldsymbol{S}$, that is to say, for the sake of subgraph anonymization, we consider every community itself as an attribute of a given community. Moreover, those communities in the column direction of a final partitioned sub-matrix are just what we call a kind of "*Support Condition*" $\boldsymbol{T}$ (target community subset) under which communities in row direction are considered as similar enough to be located in the same cluster.

Next, in step 04, Z-score method has been applied to convert similarity matrix $\boldsymbol{S}$ into a binary-value one $\boldsymbol{S}$**'** consisting only of 1 and 0 value.

More importantly, we utilize ***MCA*** algorithm to partition sub-matrix $\boldsymbol{S}''$ from $\boldsymbol{S}'$ in step 05 after previous normalization and the sub-matrix $\boldsymbol{S}''$ is just a similar-community cluster

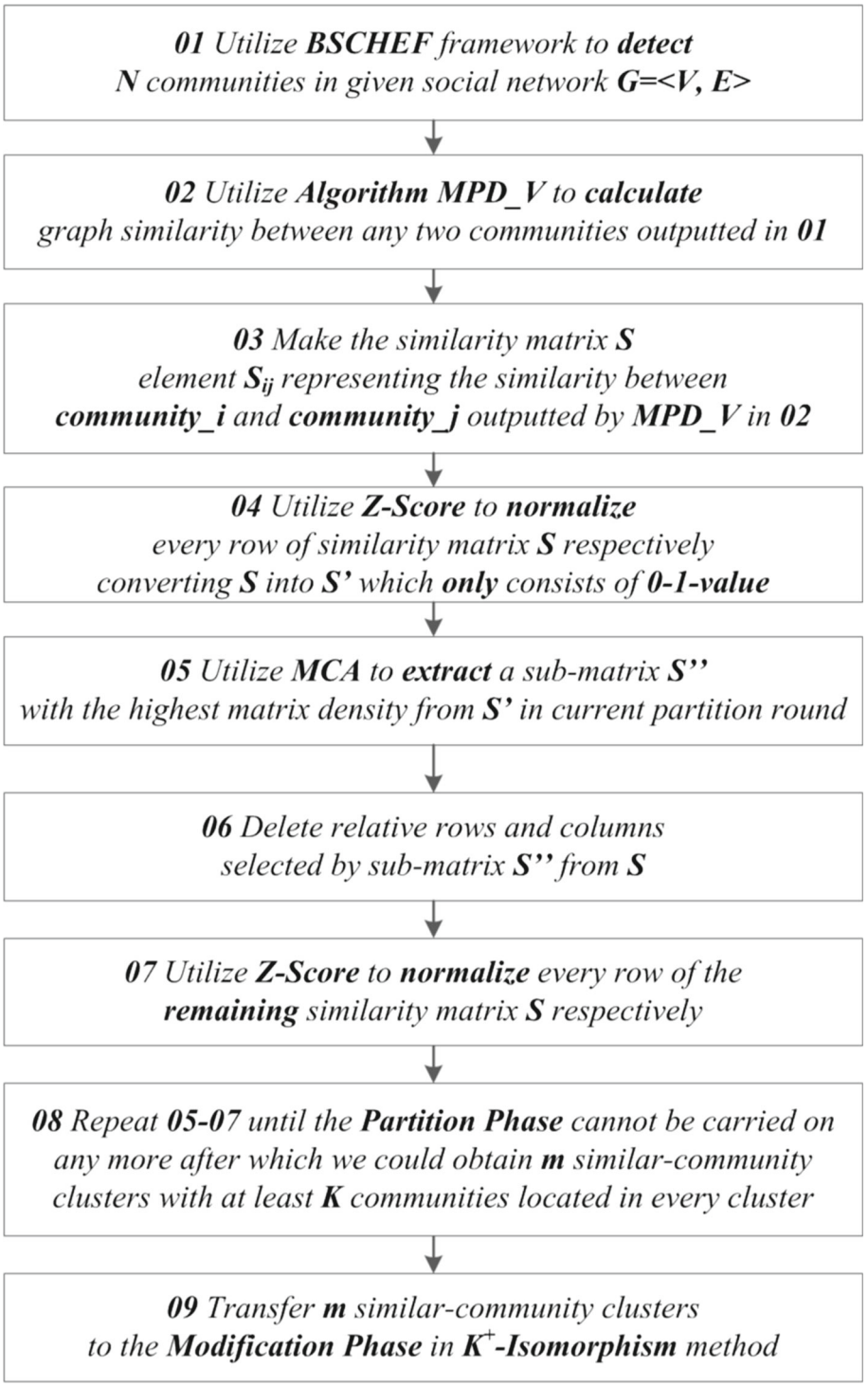

Fig. 12 Detailed process of partition phase to obtain similar-community clusters in K^+-isomorphism method

that we want to obtain in partition phase of K^+-isomorphism method. Then, the deletion of rows and columns involved in sub-matrix S'' will be conducted which means the initial similarity matrix S needs to be decomposed to a new state from which we again use Z-score method to normalize every row of the remaining similarity matrix S in step 07. Finally, repeat sub-matrix partition, rows and columns deletion for decomposition and re-normalization of the remaining similarity matrix S as step 08 until no more sub-matrix can be detected. In this way, we eventually obtain m similar-community clusters or sub-matrix.

It is worthwhile to be noticed that in the partition process of ***MCA*** shown in Algorithm 4, we have introduced a new parameter k which defines the minimum number of rows in a

Algorithm 4 ***MCA*** sub-matrix clustering algorithm based on matrix density

Require: $\boldsymbol{M}_{i*j}$: initial similarity matrix.
$\boldsymbol{k}$:the minimum number of rows in sub-matrix.
Ensure: $\boldsymbol{S}_{i*j}$: a sub-matrix with higher density than $\boldsymbol{M}_{i*j}$.
Calculate the matrix density Ψ of $\boldsymbol{M}_{i*j}$
/** The start of shrink or partitioning process **/
Set the start row number as ***row_num***
4: *Select* elements as $\boldsymbol{m}_{i,j} = \boldsymbol{1}$ in row ***row_num*** of $\boldsymbol{M}_{i*j}$ to build sub-matrix $\boldsymbol{S}_{1*j}$
/** Row Expansion **/
Select $\boldsymbol{j}$ columns whose $\boldsymbol{s}_{1,j} = \boldsymbol{1}$ in $\boldsymbol{S}_{1*j}$ as benchmark
Fill elements as $\boldsymbol{s}_{i,j} = \boldsymbol{1}$ only in $\boldsymbol{i}$ rows involved in $\boldsymbol{j}$ columns above to build sub-matrix $\boldsymbol{S}_{i*j}$
8: *Record* the position of last column of $\boldsymbol{S}_{i*j}$ in $\boldsymbol{M}_{i*j}$, marked as ***last_column***
/** Row Cut **/
Calculate the sum of every row in $\boldsymbol{S}_{i*j}$ and delete rows with the minimum sum
Update sub-matrix $\boldsymbol{S}_{i*j}$
12: **if** the number of rows $\boldsymbol{R}$ in $\boldsymbol{S}_{i*j} < \boldsymbol{k}$ **then**
delete matrix $\boldsymbol{S}_{i*j}$ and shift the ***row_num*** downwards
If no more row for shift, then output $\boldsymbol{M}_{i*j}$, turn to line 40
Otherwise, turn to line 03
16: **end if**
if current state of matrix $\boldsymbol{S}_{i*j}$ only consisting of 1 value **then**
cover $\boldsymbol{M}_{i*j}$ by $\boldsymbol{S}_{i*j}$, turn to line 40
end if
20: /** Column Expansion **/
Use last_column as benchmark;
Add all columns before ***last_column*** in $\boldsymbol{M}_{i*j}$ to $\boldsymbol{S}_{i*j}$ with constraint of involved rows in $\boldsymbol{S}_{i*j}$
Update sub-matrix $\boldsymbol{S}_{i*j}$
24: ***/** Column Cut **/***
Calculate the sum of every column in $\boldsymbol{S}_{i*j}$ and *delete* columns with the minimum sum;
Update sub-matrix $\boldsymbol{S}_{i*j}$
if current state of matrix $\boldsymbol{S}_{i*j}$ only consisting of 1 value **then**
28: cover $\boldsymbol{M}_{i*j}$ by $\boldsymbol{S}_{i*j}$, turn to line 40
end if
/** the density of empty matrix is 0 **/
if the density $\boldsymbol{D}$ of sub-matrix $\boldsymbol{S}_{i*j} > \Psi$ **then**
32: ***/** insures the initial similarity matrix $\boldsymbol{M}_{i*j}$ can be shrunk continuously **/***
cover $\boldsymbol{M}_{i*j}$ by $\boldsymbol{S}_{i*j}$, *delete* S_{i*j}, turn to line 01
else
delete $\boldsymbol{S}_{i*j}$, shift the ***row_num*** downwards;
36: If no more row for shift, then output $\boldsymbol{M}_{i*j}$, turn to line 40
Otherwise, turn to line 03
end if
/** The start of decomposition process **/
40: *Delete* rows and columns selected by final state of $\boldsymbol{S}_{i*j}$ from $\boldsymbol{M}_{i*j}$ for decomposition;
Use Z-score to re-normalize every row of the new state of $\boldsymbol{M}_{i*j}$
Finish current round of sub-matrix partition

partitioned sub-matrix. In other words, for step 05 in Fig. 12, if there are less than ***2k*** rows in the current state of similarity matrix $\boldsymbol{S}$, then the sub-matrix partition process can be stopped so that we are able to ensure there are at least $\boldsymbol{k}$ rows in all the partitioned sub-matrix or, at least $\boldsymbol{k}$ subgraphs located in every similar-community cluster.

In Algorithm 4, line 01 calculates the density Ψ of the initial similarity matrix $\boldsymbol{M}_{i*j}$ and builds sub-matrix $\boldsymbol{S}_{1*j}$ in line 04, a row vector of $\boldsymbol{M}_{i*j}$. In line 7, the algorithm builds sub-matrix $\boldsymbol{S}_{i*j}$. Both $\boldsymbol{S}_{1*j}$ and $\boldsymbol{S}_{i*j}$ are the middle state of a final partitioned sub-matrix. Lines 05–11 process the matrix $\boldsymbol{S}_{i*j}$ by the rule of "Row expansion; Row cut."

Line 12 checks whether the number of subgraphs or communities located in sub-matrix $\boldsymbol{S}_{i*j}$ is less than the value of $\boldsymbol{k}$ which means the number of row in sub-matrix $\boldsymbol{S}_{i*j}$ must be more than $\boldsymbol{k}$, otherwise, $\boldsymbol{S}_{i*j}$ will be deleted and reset to the current state of $\boldsymbol{M}_{i*j}$ for another selection of new ***row_num***. Line 17 checks whether all the elements in $\boldsymbol{S}_{i*j}$ are 1, if so, it means the density of sub-matrix $\boldsymbol{S}_{i*j}$ has already reached the highest level and the current round of shrink or partitioning can be stopped after which $\boldsymbol{M}_{i*j}$ will be covered by $\boldsymbol{S}_{i*j}$.

Lines 20–26 process sub-matrix $\boldsymbol{S}_{i*j}$ by the rule of "Column expansion; Column cut." Line 27 is the same as line 17 which checks whether the density of sub-matrix $\boldsymbol{S}_{i*j}$ has already reached the highest value, or only containing 1 value.

Lines 30–38 are the most critical part. The algorithm firstly judges whether the matrix density of $\boldsymbol{S}_{i*j}$ is higher than that of $\boldsymbol{M}_{i*j}$ after one process of row–column expansion and cut, if so, then $\boldsymbol{M}_{i*j}$ will be covered by $\boldsymbol{S}_{i*j}$ as the new initial state for further row–column expansion and cut; otherwise, $\boldsymbol{S}_{i*j}$ will be discarded and the execution of algorithm will return to $\boldsymbol{M}_{i*j}$ for new selection of ***row_nums***.

Lines 40–41 are conducted after the finish of whole shrink or partition process from lines 01 to 39. Step 40 decomposes the initial similarity matrix $\boldsymbol{M}_{i*j}$ by deleting the rows and columns selected by the final state of sub-matrix $\boldsymbol{S}_{i*j}$ after partitioning. In this way, rows and columns used by one sub-matrix or cluster will not be selected by other sub-matrix again so that we ensure the partition scheme obeys the row–column exclusive model shown in Fig. 10b. Line 41 utilizes Z-score method to re-normalize every row of the remaining similarity matrix $\boldsymbol{M}_{i*j}$ after decomposition to prevent all the value in the remaining similarity matrix $\boldsymbol{M}_{i*j}$ is 0.

4.2 Modification phase in every cluster

After partitioning all the similar-community clusters, then in the modification phase of our proposed K^+-isomorphism method, we modify the subgraphs or communities located in the same cluster to make them isomorphic to each other. We have already done the work of KDVEM in [37], a novel algorithm to achieve ***k-degree*** anonymization in social network. The KDVEM algorithm consists of ***greedy_partition*** as well as ***greedy_modification*** steps. In the modification phase of K^+-isomorphism method, we imitate KDVEM to modify all the communities located in one cluster by above two steps.

In the first step, we adjust all the communities in ***Cluster_i*** to the same size which means all the communities have the same number of vertexes and edges but their structures are

Algorithm 5 ***Greedy_shuffle*** algorithm to modify all the communities or subgraphs in a cluster to the same size

Require: ***Cluster_i***: subgraphs or communities list {C_1, C_2, C_3, ., C_n}
Ensure: A preserved ***Cluster_i***: contains ***n*** subgraphs or communities with the same size of vertexes and edges.
Select a medium size of community C' in ***Cluster***$_i$ as a ***Template***
Make a complete *tuple-list mappings* ***M*** of ***C'*** as <*vertex_id*, *degree*>
Arrange all the items in *mappings* ***M*** *of* ***C'*** in a descending order by degree
/**Start to shuffle the rest subgraphs in Cluster_i to the same size as C' **/
5: **for all** ***C_temp*** in {C_1, C_2, C_3, ., C_{n-1} } **do**
Make a complete *tuple-list mappings* ***M'*** of ***C_temp*** as <*vertex_id*, *degree*>
Arrange all the items in *mappings* ***M'*** of ***C_temp*** in a descending order by degree
for all ***item I'*** in ***M'*** **do**
/** use vertex_id to identify whether the node is an original one, or a new added one **/
10: *Extract* ***item I*** from ***M*** as well as ***item I'*** from ***M'*** in the above arranged order
if *I'.degree* == *I.degree* **then**
continue
end if
if *I'.degree* < *I.degree* **then**
15: *Add* edges between current node and the nearest unchecked nodes in ***C_temp***;
Update the *mappings* ***M'*** of ***C_temp***;
end if
if *I'.degree* > *I.degree* **then**
Delete edges randomly from the nearest unchecked nodes in ***C_temp***;
20: *Update* the *mappings* ***M'*** of ***C_temp***;
end if
if no more *items* in ***M'*** **then**
Copy the rest tuples from ***M*** to ***M'***;
Update the *mappings* ***M'*** of ***C_temp***;
25: **end if**
if no more *items* in ***M*** **then**
Delete the rest tuples in ***M'*** of ***C_temp***;;
Update the *mappings* ***M'*** of ***C_temp***;
end if
30: **end for**
/**subgraphs or communities in Cluster_i have the same size of vertex**/
Count number of ***edge*** as edges in ***template***
Count number of edge as ***edge'*** in ***C_temp***
Delete or *Add* edges from vertex with the highest degree in ***C'*** until ‖***edges***‖=‖***edges'***‖
35: ***/**subgraphs or communities in Cluster_i have the same size of edge**/***
end for

not isomorphic by using ***greedy_shuffle*** shown in Algorithm 5 inspired by the ***greedy_partition*** of KDVEM. In the process of ***greedy_shuffle***, we also select a community with the medium size as ***Template*** in ***Cluster_i***.

Then, in the second step, we make communities in ***Cluster_i*** isomorphic to the ***Template*** only by edge moving. The "edge moving" is conducted according to the traversal of a spanning tree made from two given communities or subgraphs whose principle is identical to that of ***Ullmann*** algorithm (Ullmann 1976).

For algorithm 5, in lines 1–3, we firstly select a community ***C'*** with the medium size in cluster as ***Template*** which means all the other communities will be modified to ***C'***. In line 2, mappings formatted as $<vertex\_id, degree>$ of community ***C'*** will be made and arranged in a descending order by degree in line 3.

In lines 6–30, for the rest communities in current cluster, we also make the $<vertex\_id, degree>$ mappings for ***C_temp*** in line 6 and arrange them by degree in descending order in line 7. Then, from line 10, we extract items in mappings of ***C_temp*** and ***C'*** one by one to modify the degree of existed vertex by adding or deleting edges in lines 11–21 during which the updates of mappings are always maintained. Lines 20–26 process the case if mappings of ***C_temp*** are more or less than those of ***C'*** to ensure the number of vertex in ***C_temp*** is same to that of ***C'***.

In addition, in lines 6–30, we always maintain the ***id*** or ***label*** of vertex to identify whether it is an original one, or a new added one.

Finally in lines 32–34, we modify all the communities until they have the same number of edge. We use a pair of vertex ***ids*** or ***labels*** to identify whether an edge is a new added one.

In the second step, the edge moving will be conducted. Having communities or subgraphs with same number of vertex and edge, we then adjust them to an isomorphic state only by moving edge from one pair of vertexes to the other with the help of ***Ullmann*** algorithm (Ullmann 1976). Figure 13 shows an example of edge moving. With two given subgraphs like C_1 and C_2, ***Ullmann*** algorithm traverses the spanning tree of community C_1 and C_2 in depth to judge whether C_2 is isomorphic to C_1. In Fig. 13, the edge moving algorithm starts from two dashed vertexes, checking edges in the order marked by the number and in the fourth step, when an edge cannot be found in the next level of current tree structure, we move an edge from the nearest checked part to the target position shown in C_2. The detailed steps of traversing spanning tree in ***Ullmann*** algorithm can be seen in Ullmann (1976).

Last but not the least, using ***Cluster B*** in Fig. 8 as an example, Fig. 14 demonstrates the whole process of modifying all

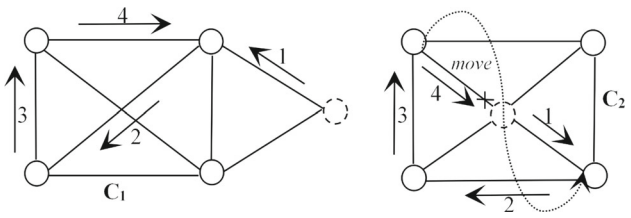

Fig. 13 An example of edge moving step based on ***Ullmann*** algorithm

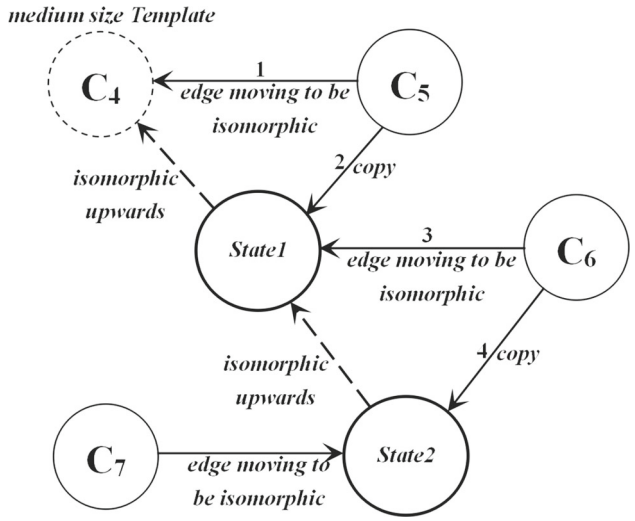

Fig. 14 Process of adjusting all same-size communities to an isomorphic state by edge moving in the second step

the same-size communities to an isomorphic state by *edge moving*, the second step of modification phase.

Firstly, by ***greedy_shuffle*** in ***Cluster B***, we can select a community ***C'*** with the medium size mentioned in line 1 of **Algorithm 5** as ***Template***, taking C_4 as an example. Then, modify any two communities by edge moving, such as C_4 and C_5, to make C_5 isomorphic to C_4 and ***state1*** is just a copy of modified C_5.

Next, using ***state1*** as a ***Template***, do edge moving between ***state1*** and C_6 to obtain an isomorphic ***state2*** which is also a copy of modified C_6. Repeat above process iteratively until all the communities are isomorphic to the current ***state***. That is to say, all the communities in ***Cluster B*** are all isomorphic to the selected medium size ***Template***, like C_4.

5 Experiments

In this section, we firstly test the effectiveness and efficiency of MPD_V compared with its prototype MPD as well as other representative algorithms for detecting the maximum common subgraph such as McGregor (Mc) (Mcgregor 1982) and Durand-Pasari (D-P) (Bunke and Shearer 1998). D-P detects the maximum common subgraph by finding the maximum clique in an association graph generated by data graph G and query graph Q. Mc detects the maximum common subgraph by back-tracking. Secondly, we compare the ***anonymization cost*** of our proposed K^+-isomorphism method with other K-isomorphism techniques when achieving the state of subgraph K-anonymization, or K-isomorphism, in their own meaning. All the experiments are conducted on computers with memory 8 GB 1600 MHz DDR3 and 2.4 GHz Intel Core i5 processor.

Table 2 Datasets used in experiments of MPD_V

Dataset name	Number of vertex	Number of edge
Karate	34	78
American college football team network	115	613
Gnutella08	630	2077

5.1 Experiments of MPD_V

In experiments of MPD_V, we select three datasets shown in Table 2. Karate dataset (JunqiuLi 2014) contains the relationship among 34 members of a karate club; American college football team network dataset (Girvan and Newman 2002) was collected by Girvan and Newman recording the relationship between American college football teams; Eustace and Wang (2015) is a simple P2P file exchanging protocol, and Gnutella08 reflects topology of file exchanging between computers on a particular occasion.

In order to test the worst situation of MPD_V, we conduct complete search in the layered-linked search space which means all the match pairs will be used as an initial match pair to trigger a new round of detection. Meanwhile, other algorithms such as MPD, Mc, D-P will also conduct complete search in their own way.

5.1.1 Efficiency of MPD_V

For different datasets, we randomly select several edges as query graphs in range of 20 and 80%, using the original dataset as data graphs. Then, for query graphs from ratio of 20–80%, we use different algorithms to process them, repeating 10 times, and calculate the average size of searched maximum common subgraphs. It is worthwhile to be noticed that the selections of query graphs of different ratios are conducted only once.

Tables 3, 4 and 5 demonstrate the average size of the maximum common subgraph searched from query graphs with different edge ratios. It can be apparently observed that MPD_V has obtained the largest size of searched common subgraphs except the case of 40% ratio in Table 4 where D-P obtained the largest size of searched common subgraphs.

In conclusion, MPD_V, our improved algorithm, has much better efficiency to search larger common subgraphs compared with its prototype and other techniques.

5.1.2 Performance of MPD_V

Tables 6, 7 and 8 list the average execution time to finish a complete search on query graphs with different ratios. Firstly, by comparing the average execution time between MPD and MPD_V as well as the last line in Tables 6, 7 and 8, it can be found that the results of two algorithms are close to each

Table 3 Average size of searched maximum common subgraph in *Karate*

The ratio of query graph	20%	40%	60%	80%
Average size of MPD	4.19	5.41	10.08	11.60
Average size of MPD_V	**5.78**	**9.27**	**17.98**	**20.69**
Average size of Mc	4.21	7.92	14.76	16.48
Average size of D-P	4.85	7.44	14.17	15.82

Table 4 Average size of searched maximum common subgraph in *Football Team Network*

The ratio of query graph	20%	40%	60%	80%
Average size of MPD	35.81	39.04	39.79	40.17
Average size of MPD_V	**36.81**	43.45	**46.82**	**47.96**
Average size of Mc	36.43	42.83	46.78	47.04
Average size of D-P	35.85	**43.99**	45.35	46.93

Table 5 Average size of searched maximum common subgraph in *Gnutella08*

The ratio of query graph	20%	40%	60%	80%
Average size of MPD	82.48	85.08	86.37	94.32
Average size of MPD_V	**85.81**	**87.62**	**91.93**	**102.05**
Average size of Mc	84.06	85.89	90.05	99.33
Average size of D-P	83.70	87.34	89.62	98.94

other which means although MPD_V has added a result set verification phase after each search round, yet the execution time of MPD_V has not increased dramatically compared to that of MPD.

Next, by picking up the highest result in all the columns presented in Tables 6, 7 and 8, we can discover that the execution time of MPD_V is also close to that of Mc as well as D-P.

More specifically, we can find that in the third column of Table 6, the last two columns in Table 7, the second and fourth columns in Table 8, the execution time of MPD_V has not reached the peak.

In conclusion, the improvement of MPD_V by adding result set verification phase has not aggravated its execution time and the performance is still within an acceptable scope compared to other techniques.

5.2 Experiment of K^+-isomorphism method

We use "***anonymization cost***" to evaluate the extent of modification, or utility for a given network before and after preservation. Equation (3) has illustrated the way to calculate ***anonymization cost*** for a given graph $\boldsymbol{G}$.

Equation 3 (Anonymization cost) *Given a simple and undirected graph* $G =< V, E >$, $G_P =< V_P, E_P >$ *is the preserved graph of G after any type of anonymization process, then the anonymization cost from G to* G_P *is shown as below which reflects the cost to add or delete edges and vertexes.*

$$cost(G,G_P) = \| \left\{E \bigcup E_P\right\} - \left\{E \bigcap E_P\right\} \| + \|V_P\| - \|V\| \tag{3}$$

The reason why we only use ***anonymization cost*** to evaluate our experiment is that our proposed K^+-isomorphism method is mainly targeted at preserving subgraphs or communities located in clusters while all the current K-isomorphism techniques preserve the whole network. As a result, indexes such as transitivity, APL and ACC checking features of the whole preserved network analyzed in Ma et al. (2015a) will not be applied to our proposed K^+-isomorphism method any more.

For our proposed K^+-isomorphism method, we have mentioned that we maintain the vertex ***id*** or label through the whole modification phase to identify whether an edge or vertex is an original one or new added one. In this way, it is feasible to use ***cost(G, G_P)*** to calculate ***anonymization cost*** (Xie et al. 2016) for every community and their sum will be the whole anonymization cost of a community set $\{\boldsymbol{C}_1, \boldsymbol{C}_2, \boldsymbol{C}_3, ., \boldsymbol{C}_n\}$ after preservation.

Fresp (Guoting et al. 2016) is an optimal K-isomorphism algorithm famous for decreasing the modification of the whole network when achieving K-isomorphism state by substituting frequent non-overlapped subgraphs iteratively. Due to the feature of preserving the whole network, we use ***cost(G, G_P)*** in a usual way to calculate the ***anonymization cost*** for ***Fresp*** when it is processed on a given network $\boldsymbol{G}$. We use BSCHEF (Ma et al. 2016b) to detect communities on a subset of C-DBLP, a kind of co-author social network in China. The feature of selected dataset is shown in Table 9.

Table 6 Average execution time on *Karate*

The ratio of query graph	20%	40%	60%	80%
Average execution time of MPD (ms)	2521	3027	12621	144,790
Average execution time of MPD_V (ms)	**2966**	**3292**	15,650	**173.748**
Average execution time of Mc (ms)	2783	3206	**16,543**	150,289
Average execution time of D-P (ms)	2701	3239	14,917	146,167
MPD_V versus MPD	+17.65%	+8.75%	+24%	+20%

Table 7 Average execution time on *Football Team Network*

The ratio of query graph	20%	40%	60%	80%
Average execution time of MPD (ms)	104,549	367,022	1,669,155	5,575,311
Average execution time of MPD_V (ms)	**126,090**	**399,223**	1,824,728	5,841,438
Average execution time of Mc (ms)	105,625	355,839	**1,868,550**	5,414,404
Average execution time of D-P (ms)	100,702	369,873	1,656,613	**6,022,697**
MPD_V versus MPD	+20.60%	+8.77%	+9.32%	+4.77%

Table 8 Average execution time on ***Gnutella08***

The ratio of query graph	20%	40%	60%	80%
Average execution time of MPD (ms)	270,032	1,435,647	5,021,173	16,877,140
Average execution time of MPD_V (ms)	**291,706**	1,550,971	**5,672,578**	17,032,796
Average execution time of Mc (ms)	239,073	**1,580,202**	5,247,721	**17,841,438**
Average execution time of D-P (ms)	261,956	1,494,738	5,405,010	16,639,709
MPD_V versus MPD	+8.03%	+8.03%	+12.08%	+0.92%

Table 9 Feature of selected subgraph and detected communities by BSCHEF in C-DBLP

Subgraph type	Simple and Undirected
Number of vertex	1000
Number of edge	2443
Modularity of community set	0.4252591037
BSCHEF parameters	$\mu = 3, \varepsilon = 0.28$
Number of communities detected by BSCHEF	59

For our proposed K^+-isomorphism method, we have introduced a new parameter $\boldsymbol{k}$ in ***MCA*** which is used in sub-matrix partitioning phase and represents the least number of communities located in every cluster, or the least number of rows located in every partitioned sub-matrix. In Fresp, it also has a parameter $\boldsymbol{K}$ set at the beginning of preservation to represent the least number of partitioned subgraphs constituting the entire network.

We set $\boldsymbol{k}$ and $\boldsymbol{K}$ from 2 to 28 (the total number of communities detected by BSCHEF is 59) and compare the ***anonymization cost*** calculated by Eq. (3) under the same value of $\boldsymbol{k}$ and $\boldsymbol{K}$ when achieving K-isomorphism state in their own meaning. The results of ***anonymization cost*** are shown in Table 10 whose tendency is shown in Fig. 15.

In order to show the feature of experiment results more directly, on the one hand, we temporarily use $\boldsymbol{K}$ to represent the $\boldsymbol{k}$ of K^+-isomorphism and the comparison of anonymization cost between our proposed K^+-isomorphism method and Fresp is made under the same value of $\boldsymbol{K}$; on the other hand, as the comparison is made for $\boldsymbol{K}$ from 2 to 28, we use the ratio in the form of $\boldsymbol{K}/community\_number$ (here 59) to represent the specific value of $\boldsymbol{K}$.

In Fig. 15, when the value of $\boldsymbol{K}$ is less than or equal to 27%, the ***anonymization cost*** of our proposed K^+-isomorphism method is smaller than that of ***Fresp*** because when using ***MCA*** to extract sub-matrix based on density, the initial similarity matrix $\boldsymbol{M}$ can be continuously shrunk if and only if the number of rows in current $\boldsymbol{M}$ is more than $\boldsymbol{K}$. The final similar-community clusters partitioned by proposed K^+-isomorphism method present the feature—more clusters with less communities located in.

As a result, for K^+-isomorphism method, the modification of every cluster is minor and the sum of modification, or ***anonymization cost*** produced by all the communities is smaller.

However, when the value of $\boldsymbol{K}$ is less than or equal to 27%, ***Fresp*** still preserves the whole network; therefore, it obtains a larger extent of modification, or higher ***anonymization cost***.

In Fig. 15, when the value of $\boldsymbol{K}$ is more than 27%, the ***anonymization cost*** of our proposed K^+-isomorphism method is more than that of Fresp because at this stage, the final similar-community clusters partitioned by proposed K^+-isomorphism method present the feature—less clusters with more communities located in. In other words, the number of communities in every cluster is close to that of subgraphs partitioned in Fresp.

As a result, for K^+-isomorphism method, the modification of every cluster is larger and the final ***anonymization cost*** is higher. However, due to the advantage of substituting frequent non-overlapped subgraphs during modification phase when preserving the whole network, Fresp obtains a lower ***anonymization cost***.

In conclusion, when it comes to preserve the subgraphs or communities in a social network, our proposed K^+-isomorphism method can achieve the K-isomorphism state with less modification of original network structure, or lower

Table 10 Comparison of *anonymization cost* between proposed K^+-*isomorphism* method and *Fresp*

Anonymization cost	$K = 3\%$	$K = 6\%$	$K = 10\%$	$K = 13\%$	$K = 17\%$	$K = 20\%$	$K = 24\%$
K^+-*isomorphism*	**96**	**57**	**131**	**215**	**187**	**259**	**318**
Fresp	204	236	323	357	414	448	475
Anonymization cost	$K = 27\%$	$K = 31\%$	$K = 34\%$	$K = 37\%$	$K = 41\%$	$K = 44\%$	$K = 47\%$
K^+-*isomorphism*	**383**	563	557	624	842	976	1025
Fresp	461	**537**	**519**	**596**	**638**	**713**	**802**

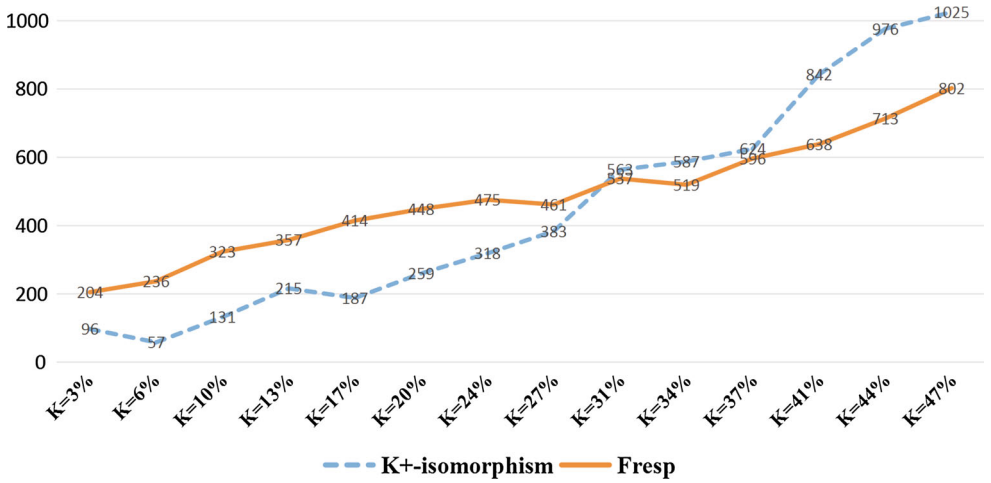

Fig. 15 Tendency of ***anonymization cost*** presented in Table 10

anonymization cost compared to the current K-isomorphism method.

6 Conclusions and future works

In this paper, we proposed a novel K^+-isomorphism method to achieve the state of K-anonymization for subgraphs or detected communities in a social network. The K^+-isomorphism method consists of partition phase and modification phase. In partition phase, we firstly use the principle of "bi-clustering" and its matrix version to define the problem of ***n*** similar-subgraph cluster partition. Then, with the help of ***MCA*** algorithm, we detect sub-matrix on the similarity matrix ***S*** built by our improved maximum common subgraph detection algorithm MPD_V and every sub-matrix is just a similar-subgraph cluster which we want to obtain in partition phase of K^+-isomorphism method. In modification phase, we imitate our work of KDVEM to modify all the subgraphs or communities located in every cluster to a mutually isomorphic state by steps of ***greedy_shuffle*** and edge moving.

However, there still exist some unexplored points of our work. In the future, we will pay more attention to increase the effectiveness and efficiency of K^+-isomorphism method. For one thing, the parallelism of K^+-isomorphism method is still a blank field of K-anonymization preservation in social network. For another, more efforts need to be devoted in order to strengthen the ability of K^+-isomorphism method to resist malicious attacks.

Acknowledgements This work was supported in part by National Science Foundation of China (No. 61572259), Special Public Sector Research Program of China (No. GYHY201506080), and was also supported by PAPD.

Compliance with ethical standards

Conflict of interest Tinghui Ma has received research Grants from National Science Foundation of China and Special Public Sector Research Program of China.

Informed consent Informed consent was obtained from all individual participants included in the study.

References

Bhagat S, Cormode G, Krishnamurthy B, Srivastava D (2010) Prediction promotes privacy in dynamic social networks. In: Proceedings of the 3rd conference on online social networks, pp 6–6

Bin G, Sheng VS, Wang Z, Ho D, Osman S, Li S (2015a) Incremental learning for v-support vector regression. Neural Netw 67:140–150

Bin G, Sheng VS, Tay KY, Romano W, Li S (2015b) Incremental support vector learning for ordinal regression. IEEE Trans Neural Netw Learn Syst 26(7):1403–1416

Bunke H, Shearer K (1998) A graph distance metric based on the maximal common subgraph. Pattern Recognit Lett 19(3–4):255–259
Casas-Roma J, Herrera-Joancomart J, Torra V (2014) Anonymizing graphs: measuring quality for clustering. Knowl Inf Syst 44(3):507–528
Chakravorty A, Wlodarczyk TW, Rong C (2014) A scalable K-anonymization solution for preserving privacy in an aging-in-place welfare intercloud. IEEE international conference on cloud engineering. IEEE, pp 424–431
Chen B, Shu H, Coatrieux G, Chen G, Sun X, Coatrieux J-L (2015) Color image analysis by quaternion-type moments. J Math Imaging Vis 51(1):124–144
Cheng J, Fu W C, Liu J (2010) K-isomorphism: privacy preserving network publication against structural attacks. In: ACM SIGMOD international conference on management of data, SIGMOD 2010, Indianapolis, Indiana, USA, June, pp 459–470
Doka K, Xue M, Tsoumakos D et al (2015) k-Anonymization by freeform generalization. In: ACM symposium on information, computer and communications security. ACM, pp 519–530
Eustace J, Wang X (2015) Overlapping community detection using neighborhood ratio matrix. Phys A 421:510–521
Girvan M, Newman MEJ (2002) Community structure in social and biological networks. Proc Natl Acad Sci 99(12):7821–7826
Gouda K, Hassaan M (2012) A fast algorithm for subgraph search problem. In: 2012 8th international conference on informatics and systems (INFOS). IEEE, pp DE-53–DE-59
Guo P, Wang J, Li B, Lee S (2014) A variable threshold-value authentication architecture for wireless mesh networks. J Internet Technol 15(6):929–936
Guoting F, Yonglong L, Dandan S, Taochun W, Xiaoyao Z (2016) Edge partitioning approach for protecting sensitive relationships in social network. J Comput Appl 36(1):207–211
Huda MN, Yamada S, Sonehara N (2013) An efficient k-anonymization algorithm with low information loss. Lect Notes Electr Eng 156:249–254
Hu H, Li G, Feng J (2013) Fast similar subgraph search with maximum common connected subgraph constraints. In: IEEE International Congress on big data, pp 181–188
Jiang J, Tan Z, Zhao N et al (2010) MultiMarker propagation Web log mining algorithms based on weighted matrix cluster. In: 2010 2nd international conference on future computer and communication (ICFCC). IEEE, pp V2-374–V2-378
Kiyomoto S, Miyake Y (2014) How to find an appropriate K for K-anonymization. In: 2014 eighth international conference on innovative mobile and Internet services in ubiquitous computing (IMIS). IEEE, pp 273–279
Kuo JJ, Chang JS, Zhang YJ (2013) Visualized book recommender system using matrix clustering. J Educ Media Libr Sci 51(1):5–35
Li J, Wang X (2014) Uncovering the overlapping community structure of complex networks by maximal cliques. Phys A 415:398–406
Liu X, Yang X (2011) A generalization based approach for anonymizing weighted social network graphs. In: International conference on web-age information management. Springer, Berlin, pp 118–130
Lv Y, Ma T, Tang M et al (2016) An efficient and scalable density-based clustering algorithm for datasets with complex structures. Neurocomputing 171:9–22
Ma T, Zhang Y, Cao J et al (2015a) KDVEM: a k-degree anonymity with vertex and edge modification algorithm. Computing 97(12):1165–1184
Ma T, Zhou J, Tang M, Tian Y, Al-Dhelaan A, Al-Rodhaan M, Lee S (2015b) Social network and tag sources based augmenting collaborative recommender system. IEICE Trans 98(D(4)):902–910
Ma T, Wang Y, Tang M et al (2016a) LED: a fast overlapping communities detection algorithm based on structural clustering. Neurocomputing 207:488–500
Ma T, Rong H, Ying C, Tian Y, Al-Dhelaan A, Al-Rodhaan M (2016b) Detect structural-connected communities based on BSCHEF in C-DBLP. Concurr Comput Pract Exp 28(2):311–330
Madeira SC, Oliveira AL (2004) Biclustering algorithms for biological data analysis: a survey. IEEE ACM Trans Comput Biol Bioinform 1(1):24–45
Mallek S, Boukhris I, Elouedi Z (2015) Community detection for graph-based similarity: application to protein binding pockets classification. Pattern Recognit Lett 62:49–54
Mcgregor JJ (1982) Backtrack search algorithms and the maximal common subgraph problem. Softw Pract Exp 12(1):23–34
Okada R, Watanabe C, Kitagawa H (2014) A k-Anonymization algorithm on social network data that reduces distances between nodes. In: Proceedings of the 2014 IEEE 33rd international symposium on reliable distributed systems workshops. IEEE Computer Society, pp 76–81
Praveena A, Smys S (2016) Anonymization in social networks: a survey on the issues of data privacy in social network sites. Int J Eng Comput Sci 5(3):15912–15918
Rajaei M, Haghjoo MS, Miyaneh EK (2015) Ambiguity in social network data for presence, sensitive-attribute, degree and relationship privacy protection. PLoS ONE 10(6):e0130693
Raymond JW, Willett P (2002) Maximum common subgraph isomorphism algorithms for the matching of chemical structures. J Comput Aided Mol Des 16(7):521–533
Ullmann JR (1976) An algorithm for subgraph isomorphism. J ACM 23(23):31–42
Wang Y, Maple C (2005) A novel efficient algorithm for determining maximum common subgraphs. In: International conference on information visualisation. IEEE Computer Society, pp 657–663
Wang Y, Xie L, Zheng B et al (2014) High utility K-anonymization for social network publishing. Knowl Inf Syst 41(3):697–725
Wen X, Shao L, Xue Y, Wei F (2015) A rapid learning algorithm for vehicle classification. Inf Sci 295(1):395–406
Wu W, Xiao Y, Wang W et al (2010) k-symmetry model for identity anonymization in social networks. In: International conference on extending database technology. ACM, pp 111–122
Xie Y, Zheng M, Liu L (2016) A personalized sensitive label-preserving model and algorithm based on utility in social network data publishing. LNCS 9567:1–11
Xu X, Numao M (2015) An efficient generalized clustering method for achieving K-anonymization. In: Third international symposium on computing and networking. IEEE Computer Society, pp 499–502
Yang J, Wang B, Yang X et al (2014) A secure K-automorphism privacy preserving approach with high data utility in social networks. Secur Commun Netw 7(9):1399–1411
Zaghian A, Bagheri A (2016) A combined model of clustering and classification methods for preserving privacy in social networks against inference and neighborhood attacks. Int J Secur Appl 10(1):95–102
Zhang ZY, Li T, Ding C et al (2010) Binary matrix factorization for analyzing gene expression data. Data Min Knowl Discov 20(1):28–52
Zhang XL, Tang Y (2014) Protecting encrypted data against inference attacks in outsourced databases. Appl Mech Mater 571–572:621–625
Zheng Y, Jeon B, Danhua X, Jonathan Wu QM, Zhang H (2015) Image segmentation by generalized hierarchical fuzzy C-means algorithm. J Intell Fuzzy Syst 28(2):961–973