A parallel self-organizing community detection algorithm based on swarm intelligence for large scale complex networks

A Parallel Self-Organizing Community Detection Algorithm Based on Swarm Intelligence for Large Scale Complex Networks

Hanlin Sun*, Wei Jie†, Christian Sauer†, Sugang Ma*, Gang Han‡, Zhongmin Wang*, and Kui Xing*
*Big Data Research Centre, Xi'an University of Posts and Telecommunications, China
†School of Computing and Engineering, University of West London, UK
‡Department of Electronic Science and Technology, Northwestern Polytechnical University, China

***Abstract*—Community detection is a critical task for complex network analysis. It helps us to understand the properties of the system that a complex network represents and has significance to a wide range of applications. Nowadays, the challenges faced by community detection algorithms include overlapping community structure detection, large scale network analysis, dynamic changing of analyzed network topology and many more. In this paper a self-organizing community detection algorithm, based on the idea of swarm intelligence, was proposed and its parallel algorithm was designed on Giraph++ which is a semi-asynchronous parallel graph computation framework running on distributed environment. In the algorithm, a network of large size is firstly divided into a number of small sub-networks. Then, each sub-network is modeled as a self-evolving swarm intelligence sub-system, while each vertex within the sub-network acts iteratively to join into or leave from communities based on a set of predefined vertex action rules. Meanwhile, the local communities of a sub-network are sent to other sub-networks to make their members have a chance to join into, therefore connecting these self-evolving swarm intelligence sub-systems together as a whole, large and evolving, system. The vertex actions during evolution of a sub-network are sent as well to keep multiple community replicas being consistent. Thus network communication efficiency has a great impact on the algorithm' performance. While there is no vertex changing in its belonging communities anymore, an optimal community structure of the whole network will have emerged as a result. In the algorithm it is natural that a vertex can join into multiple communities simultaneously, thus can be used for overlapping community detection. The algorithm deals with vertex and edge adding or deleting in the same way as the algorithm running, therefore inherently supports dynamic network analysis. The algorithm can be used for the analysis of large scale networks with its parallel version running on distributed environment. A variety of experiments conducted on synthesized networks have shown that the proposed algorithm can effectively detect community structures and its performance is much better than certain popular community detection algorithms.**

I. Introduction

Nowadays, there are a lot of complex systems involved in people's daily life. The World Wide Web, mobile communication networks, online social networks, power grids, traffic road networks, etc., are just some examples. Such a system is often modeled as a mathematical complex network to investigate. A complex network usually shows some interesting properties such as high network transitivity, power-law degree distribution, small world, scale free, the existence of community structures, and much more. The study of community structures can help us to understand those systems at a middle-scope level, just between the macroscopic level in which the whole system is considered and the microscopic level in which each node is analyzed individually. Actually the analysis of community structures has significance to many applications. For example, community structure analysis can be used in a social community or network (e.g. Facebook) which presents relationships between community members. The analysis of such networks will help to design reliable friend recommendation systems. As another example, community structure analysis can be used in detecting communities of customers with similar purchasing interest in e-business networks. This can lead to setting up efficient product recommendation systems and thus improving business opportunities for product retailers.

There is no a unanimous definition of community, however. An exact definition depends on the underlying problem and its application. For example, the definition could be based on degrees of vertexes [1], k-cliques [2], k-clans [2], k-clubs [2], etc. Filippo et.al. [1] gave out the definition of strong sense community and weak sense community according to member connection strength. Michele et.al. [3] proposed a number of meta definitions. Intuitively, a community is a group of vertices in a network that has more edges (connections) among its members but comparatively has less edges between its members and the rest of the network nodes. This simple concept is the core of nearly all community definitions.

Community structure analysis of complex networks has attracted much interest. A number of algorithms originating from different fields, such as physics, statistics, data mining, evolutionary computation and many more have been proposed. There are many different strategies behind these community detection algorithms, such as divisive hierarchy, agglomerative hierarchy, random walking, information diffusion, spectrum analysis, statistical inference and so on. Several comprehensive reviews of these methods have been conducted, for example, a survey of community discovery methods was provided with a special focus on techniques designed by statistical physicists [4]. The meta definitions of a community in a complex network was given and the majority community discovery methods was summed up based on their own definitions [3]. Overlapping community structure analysis algorithms were reviewed in [5] and [6], where those for social network analysis were reviewed in [7]. In an overlapping community structure, a vertex could join into multiple communities simultaneously, thus having an effect on an algorithm' performance. The performance of some

algorithms were compared in [6] and [8].

The properties of very complex networks induce three staple challenges for a community detection algorithm: (1) overlapping community structure detection, especially high overlapping density (a large percent of vertices are overlapping) and high overlapping diversity (an overlapping vertex belongs to a large number of communities), (2) large scale network analysis, e.g., the number of vertices and edges could reach the scale of several billions and even more, and (3) dynamic changing of the analyzed networks topology, i.e., a large number of vertices and edges could appear or disappear frequently. The problem about very dynamic, large scale networks is how to find community structures within them quickly with as less effort as possible. Most of the traditional algorithms are incapable of beating these challenges mainly because of their heavy computational cost or model limitations.

In this paper we present a Paralel Self-Organizing Community Detection (PSOCD) algorithm based on the idea of swarm intelligence (SI), which can run in distributed environment. Swarm intelligence is the collective behavior of decentralized and self-organized systems, either natural or artificial. Generally, in an SI system, there is a large number of simple individuals who can only perform simple actions and interact with nearby neighbors locally as well as with the environment. Intelligence emerges as a consequence of the sum of these simple actions and interactions. It is believed that SI seems not to be a coincidence but rather a property of a variety of systems. In the PSOCD algorithm a network to be analyzed is modeled as a SI system in which each vertex decides its own actions, i.e., leaving its original communities or joining into new communities, depending on predefined action rules. Ultimately an optimal community structure will emerge as each node acts iteratively until no node changes its community anymore. In addition a vertex is naturally allowed to join into multiple communities, thus overlapping community structure could be discovered. The approach of the PSOCD algorithm inherently supports efficient dynamic network analysis, as only community associated with vertices that are affected by network changes need to be adjusted in the way the algorithm works and no further special steps need to be considered. Lastly, the implementation of the parallel algorithm assures it could be used for large scale network analysis.

The remainder of this paper is structured as follows: In section II, some most related overlapping community detection algorithms are briefly reviewed. In section III, the PSOCD algorithm is described formally. The time complexity of the algorithm is analyzed as well. In section IV, the evaluation results of the algorithm is presented for a number of synthesized complex networks. Finally, section V concludes the paper.

II. Related works

As for overlapping community detection, generally there are two classes of algorithms: (1) algorithms that uncover overlapping communities directly on complex networks and (2) algorithms that work based on non-overlapping algorithms, i.e., first finding non-overlapping community structures by using existing non-overlapping community detection algorithms and then adjusting marginal members of discovered communities and their neighbors to discover an overlapping structure. In this section, we briefly introduce some algorithms most related to our work.

A. LPA

A label propagation algorithm (LPA) is currently the fastest algorithm for community structure analysis, with a near-linear time complexity. The idea is that, as information propagates on a network with a community structure, it will has a high probability propagating within a community. At first, each vertex is assigned a label, indicating the community to which it belongs, then each vertex sends its label to its neighbors and selects a label received from neighbors, e.g., the label adopted by the most neighbors, as its new label. By iteratively propagating labels among neighboring vertexes, the community structure will gradually emerge. Assuming that a vertex is able to hold more than one label, LPA can be extended for overlapping community detection. There are a number of improved algorithms based on the LPA approach for overlapping community detection, such as COPRA(Community Overlap PRopagation Algorithm) [9], MLPA(Multi-Label Propagation Algorithm) [10], BMLPA(Balanced MLPA) [11], SLPA(Speaker-Listener LPA) [12], LPAcw(LPA with consensus weight) [13], DLPA(Dominant LPA) [14], etc. They are differentiated by the way the selecting labels and their propagating strategies. However, the problem of LPA is that its result is unstable and the precision of correctly detected communities is unsatisfactory, due to the inner randomness of the algorithm, especially for networks with high overlapping density and high overlapping diversity.

B. Local Expansion

In local expansion algorithms a seed vertex or vertices-set is given first as an initial community, and neighboring vertices of members join into the seed community if their joining can improve the community quality. When there is no vertex left to improve the community quality by joining the community, a complete community is discovered. A new seed (vertex or vertices-set) is then selected from those vertexes not joined into any community yet and a new community is detected in the same way as the previously detected communities. Given a seed community, if all other vertexes can try to join into it, but not only those not joined into any community yet, this strategy can be used for the detection of overlapping communities. The algorithms IS(Iterative Scan) [15], RaRe(Rank Removal) [15], IS^2(Improved IS) [16], LFM [17], fast LFM [18], DOCS(Detecting Overlapping Community Structures) [19], MOSES(Model-based Overlapping Seed ExpanSion) [20], and OSLOM(Order Statistics Local Optimization Method) [21] are just some examples. However, the results of such an algorithm heavily depends on the quality of the selected seeds.

C. Local Optimization

The basic idea behind local optimization strategy is merging partial sub-communities. First, find partial communities for each vertex on its local sub-network, which is usually a sub-network called egonet, consisting of the vertex and its neighbors and their connections. Then, merge the found partial communities according to a set of given rules. For example, if two partial communities are most similar, or if the quality of the merged community is improved, then retain the

new merged community and delete the two previous, smaller, communities. The merging continues until there are no such two communities still existing. Detecting partial communities on a local egonet is a simple task since the network size is usually small, but the merging procedure is complex for there are usually a large number of partial communities needed to be tested for merging. The algorithms DEMON(Democratic Estimate of the Modular Organization of a Network) [22], PCMA(Partial Community Merger Algorithm) [23], EgoClustering [24] and the one proposed in [25] are such algorithms following the described approach of merging partial subcommunities.

D. Game Theory

The algorithms based on game theory try to mimic a procedure through which the community structure of a network developed to the current state. In such an algorithm, a vertex is viewed as a rational or selfish individual, and it decides its own community association according to a defined utility function. While the utility function is linear locally, the Nash equilibrium, at which state no individual can increase its utility, could be reached, and in such an equilibrium state, a community structure could be deduced. An individual is allowed to join into multiple communities, and thus the algorithm could be used for overlapping community detection. The algorithms in [26] and [27], PSGMAE(Pearson correlation GAME) [28], NGGAME(Neighborhood similarity GAME) [28], and SID(Social Information Diffusion) [29]are just some examples based on the game theory. Their differences lie within the design of the utility functions they employ. However, the required local linear property of the utility function may limit the usage of such a method while the community properties could not be expressed in such a function.

E. Swarm Intelligence

Swarm intelligence ranges over a number of algorithms inspired by natural bio-systems, including the Genetic Algorithm (GA) and the Ant Colony Optimization (ACO) algorithm. Both of the two algorithms were used for community detection. The GA-NET+ [30] is GA based and used for overlapping community detection by working on a link network translated from a normal network, in which a vertex and an edge represent an edge and a vertex of the original network respectively. However, due to the limitation of the representation method for evolutionary individuals, the GA based algorithm can not be used for large scale network analysis. Generally, the number of edges of a network is much larger than the number of its vertexes, therefore the size of a link network is much larger than the size of the corresponding original network. In [31], the AntCBO algorithm was proposed for overlapping community detection. Intrinsically, this algorithm is a type of LPA but the label propagation among neighboring vertices is realized by the "'ants"' of the ACO algorithm framework.

Based on the EgoClustering algorithm, the same authors proposed an overlapping community detection algorithm, the COGS (Community Optimization Graph Swarm) [32]. In COGS, a network is treated as a swarm intelligence system and a vertex interacts with its neighbors to find the Friendship-Group (a type of partial community). Then the algorithm finds Friendship-Groups that should be merged, based on the LPA idea, through propagating community labels among such groups. In COGS, the SI is only used for finding Friendship-Groups, which is a part of the community detection procedure. Our algorithm is different in that way that we use SI as a framework to imitate a procedure through which the community structure of a network reaches its current state. The authors also presented a multi-thread parallel version COGS [33].

F. Algorithms Based on Non-Overlapping Algorithms

Tanmoy [34] analyzed the community structures of a number of real networks and found that there is a strong similarity between the structure discovered by a non-overlapping community detection algorithm, after removing overlapping vertices, and the real community structure after removing overlapping vertices. Based on this finding, the author proposed the algorithm POVC (Permanence based Vertex-replication algorithm for Overlapping Community detection), in which a non-overlapping community structure is first extracted by a traditional community detection algorithm, and then the community association of marginal vertices of the discovered structure are adjusted according to the defined "Permanence based Vertex-replication" index to find overlapping structures. The algorithm presented by [35] is similar but the adjusting of marginal vertices and their neighbors is determined by the condition if their leaving or joining could improve the quality of the related community. The performance of such an algorithm is heavily dependent on whether the used non-overlapping community detection algorithm could correctly find the boundaries of communities. Moreover, the strategy of adjusting only once may omit mutual influence between sequential adjustments and thus affects the performance.

In [36] Tanmoy et. al. proposed the MeDoc (Meta clustering based Disjoint and Overlapping Community detection) algorithm that can be used for revealing both disjoint and overlapping community structure in networks. The algorithm uses an ensemble approach, i.e., combining the strength of several non-overlapping community detection algorithms thus to detect a beter community structure. However, a vertex has a chance to be attributed to correct communities only if these correct communities could be found by any of the used non-overlapping detection algorithms in any run.

III. Algorithm description

We designed the PSOCD algorithm based on the Giraph++ system [37], which is a distributed graph processing system and provides a "think like a graph" programming model.

A. Overview

Our proposed community detection algorithm consists of tree phases, namely, partition, initialization and evolution. In the first phase, partition, an analyzed large scale network is divided into a number of smaller sized sub-networks, which will be processed in parallel in the subsequent phases. In the initialization phase, the algorithm finds an initial community for each vertex of each sub-network. These initial communities provide a foundation for the following phase of community evolution. In the evolution phase the algorithm evolves the community structure by joining community associations of each vertex iteratively. After a number of evolutions, an

optimal community structure of the network will gradually emerge.

B. Partition Phase

In the partition phase a large scale network is divided into a number of smaller sub-networks with approximately the same size. Furthermore, the connections among the identified different sub-networks should be minimal. The target of equally sized sub-networks rougly equalizes the processing time for each sub-network, thus reducing idle waiting time (caused by the parallel approach of our algorithm, before the next step of the algorithm begins). Minimal connections among the identified sub-networks is aimed for because the connections between the sub-networks have a significant influence on the performance of the subsequent phase of community evolution. For example, during its evolution, a vertex should notify its neighbors of changes of its communities and its actions. These local neighbors could be notified easily in the Giraph++ system but if a neighbor locates in another sub-network, the vertex must send community change- or action-messages across a larger communication network. Therefore, a vertex is preferred to be assigned to a sub-network with as less connections with other sub-networks (i.e., less external neighbors) as possible. Currently, the "Metis" algorithm [38] is used as the partition method, which could produce a partition structure with minimum edges across partitions. The generated sub-networks are loaded into worker nodes of a Giraph++ system for parallel processing.

C. Initialization Phase

As mentioned earlier, member vertices of a community have denser connections among them, but comparatively less connections with members of other communities. There is no more denser connected part in a network than a k-clique, of which each member is connected to all the other members of the k-clique. In other words, a maximum k-clique represents the strongest sense of community. In our algorithm, we take a smallest k-clique, 3-clique, as the initial core community for the three member vertexes. For simplicity, we find a 3-clique for an uninitialized vertex of a sub-network with its two uninitialized local neighbors as their initial community. If such a 3-clique does not exist for a vertex, then the vertex forms a community with only itself as a member. The initializing algorithm for a sub-network is shown in Fig. 1. Please note that the algorithm is executed in parallel by different worker nodes responsible for different sub-networks.

To find if a 3-clique with two neighbors exists, a vertex must be able to check if its two neighbors are connected mutually to each other. Hence a vertex must know its neighbors' neighbors while executing the initializing algorithm. This is achieved by each vertex sending its neighbors to all its neighbors firstly. The sending procedure is shown in Fig. 2.

In the later evolution phase, a vertex will check each of its neighbor if it should join into the neighbor's communities. In the distributed approach, a vertex should access its external neighbors' joining communities too. For the sake of such accesses, the initialized local communities of vertices in a sub-network should be sent to their external neighbors, located in other sub-networks, to make them have a chance to join, i.e., a community could be expanded freely across sub-networks. The community sending algorithm is shown in Fig. 3.

```
1:
2:  ALGORITHM: InitializeCommunities(sub-network)
3:
4:  for (each vertex V in the sub-network) do
5:     if (V has been initialized) then
6:        continue
7:     end if
8:     if (V and its two uninitialized neighbors N_1 and N_2
       form a 3-clique) then
9:        create a new community NC containing V, N_1 and
          N_2;
10:    else
11:       create a singleton community NC containing V;
12:    end if
13:    save NC in local community structure LCS;
14:    save ID of NC in each member;
15:    tag members of NC as being initialized;
16: end for
17:
```

Fig. 1. Finding an initial community for each vertex of a sub-network.

```
1:
2:  ALGORITHM: NotifyVertexNeighbors(sub-network)
3:
4:  for (each vertex V in the sub-network) do
5:     for (each neighbor N of V) do
6:        if (N is a local neighbor) then
7:           get local neighbor N;
8:           directly set V's neighbors for N;
9:        else
10:          SendMsg(N,[operation=notifyNeighbors, V and its
             neighbors]);
11:       end if
12:    end for
13: end for
14:
```

Fig. 2. Sending neighbors of vertexes in a sub-network to all their neighbors.

D. Evolution Phase

The evolution phase plays a key role in finding an optimal community structure for a network. The main question in evolution phase is whether a vertex should join into a community, i.e., under which condition a vertex will join into a community? To address it, we define a new type of connection strength of a vertex with a community.

1) Connection Strength: Tanmoy et. al. [39] claimed that the connection strength of a vertex belonging to a community is determined by two factors: (1) the number of connections between the vertex and each other community, but not the total number of connections between the vertex and other communities and (2) the strength with which this vertex connects to the candidate community, thus not only the number of connections between the vertex and the candidate community. The strength is measured as the clustering coefficient of the vertex's neighbors belonging to the candidate community. We propose a new type of connection strength, the connection

```
ALGORITHM: NotifyNewCommunities(sub-network)

for (each community C in local community structure LCS of the sub-network) do
  for (each recently joining vertex JV of C) do
    for (each external neighbor EN of JV) do
      if (EN is not a member of C) then
        SendMsg(EN,[operation=notifyCom,C]);
      end if
    end for
  end for
end for
```

Fig. 3. Sending local communities of vertexes in a sub-network to their neighbors located in other sub-networks.

score (CS), following the two mentioned factors:

$$CS(v) = \left[\frac{I(v)}{D(v)}\right]^{(1-c_{in}(v))} \quad (1)$$

where $I(v)$ is the connection number between vertex v and a candidate joining community, $D(v)$ indicates the degree of vertex v, and $c_{in}(v)$ represents the clustering coefficient of v's neighbors belonging to the candidate community. The $D(v)$ could be replaced with $E_{max}(v)$, the maximum number of connections between v and communities, as well. In CS, a connection strength is first measured by the connection number, and then magnified by the corresponding clustering coefficient. Therefore, the CS may get a subtler distinction between connection strengths.

2) Vertex Community Evolving: A vertex either joins into communities to which some of its neighbors belong or stays as a singleton community. Therefore, in each evolution round, the algorithm updates a vertex's communities as follows: first, get currently joining communities of all neighbors of the vertex as candidate communities, which the vertex will try to join into. Note that two neighbors may join into the same community, thus the duplicated ones should be removed from the candidates for efficiency. Second, compute the connection strengths for the vertex with each candidate community. Finally, add the vertex into the candidate communities with "stronger" strength, and remove the vertex from originally joining communities which are not joined in this iteration of the evolution. The strength of which the ratio between it and the maximum connection strength exceeds a given threshold is considered as stronger. It must be noticed that only if the connection number of a vertex with a community is larger than or equal to 3, could the connection strength be computed, because the clustering coefficient is meaningful only if this condition is satisfied. If the connection number equals 2, a vertex will join into such a candidate community only if its maximum connection is not more than 3. Otherwise a vertex will not join into a candidate community. The vertex community evolving algorithm is described as in Fig. 4.

3) Keeping Consistence: Note that the algorithm evolves a vertex' communities in a distributed manner, thus the changes of a community, i.e., some vertexes joining and some leaving, should be notified to its replicas in other sub-networks in order to keep them being consistent. The algorithm achieves this by sending information on recently joining and leaving vertexes of a community to its external members, who will be responsible for updating their local replicas once all vertices of a sub-network finished evolution. The algorithm of sending information on recently joining and leaving vertexes of a community is shown in Fig. 5.

```
ALGORITHM: EvolveCommunities(sub-network)

for (each vertex V in the sub-network) do
  get joining communities of V's neighbors as candidates;
  compute connection strength CS for V with each candidate;
  for (each candidate community C) do
    if (connection number CN of V with C ≥ 3) then
      if ((CS with C / CS_max) ≥ threshold) then
        V joins into C;
      end if
    else if ((CN with C == 2) and (CN_max ≤ 3)) then
      V joins into C;
    end if
  end for
  for (each community C V joins in originally) do
    if (V does not join in C now) then
      V leaves from C;
    end if
  end for
end for
```

Fig. 4. Evolving communities of vertexes in a sub-network.

```
ALGORITHM: NotifyVertexActions(sub-network)

for (each community C in local community structure LCS) do
  for (each external member EM of C) do
    SendMsg(EM,[operation=joining, ID of C, joining vertexes and their neighbors]);
    SendMsg(EM,[operation=leaving, ID of C, leaving vertexes]);
  end for
end for
```

Fig. 5. Sending vertex actions of communities in a sub-network.

4) Community Merging: We will explain later that in our algorithm a community is just saved once in the local community structure of a sub-network and the joining local vertices of a community reference the community by saving its ID. It could happen that a community is contained in another one completely. These contained communities should be merged (and deleted) to eliminate unnecessary vertex actions and thereby speed up the algorithm' execution. Our algorithm merges communities from a vertex viewpoint, i.e., it checks each community a vertex belongs to if it is contained by another community the vertex joining. The reason is that if a community is contained by another one, the common members

```
ALGORITHM: MergeCommunities(sub-network)

for (each vertex V in the sub-network) do
    get V's joining communities;
    sort communities firstly by their sizes, then by community IDs;
    for each small community SC do
        if SC equals another community then
            delete the community with small ID from LCS;
            notify local members of the deleted community;
        else if SC is contained by a big community BC then
            delete SC from LCS;
            notify local members of the deleted community;
        end if
    end for
end for
```

Fig. 6. Merging communities from a vertex viewpoint.

must join into both, and thus the merging could be found easily by searching within a common member's joining communities. This search strategy greatly reduces the effort needed to find a containing community for a given community. The merging algorithm is shown in Fig. 6.

While deleting a community, the deletion should be notified to members of the community. Fortunately, if a merging occurs, only local members of the community need to be notified, but external members located in other sub-networks need not. It is because that if a deleted community exists in a sub-network, the containing community must exist too. Therefore, the same merging will occur as well. However, there is a special case needed to implement the same merging occurring in different sub-networks. This special case is the situation that the contained and containing communities are completely the same. It should be guaranteed that in such a situation the deleted community should be the same one. In our implementation, we fulfill this requirement by always deleting the community with a small ID.

E. Alogrithm Framework

The framework of our proposed algorithm is show in Fig. 8. The functions start with "process" process messages received from last super-step for vertices in a sub-network and update states of communities in local community structures and the states of local vertices. In super-step 0, the algorithm notifies neighbors of each local vertex' neighbors. In super-step 1, after processing neighbor notification messages, the algorithm finds an initial community for each local vertex and notifies its external neighbors the initial community. The two functions "NotifyNewCommunities" and "NotifyVertex-Communities" (described in Fig. 7) together complete such notifications. In subsequent even super-steps, the algorithm first processes received community notification messages, then evolves communities of each vertex in a sub-network and finally is sending vertex actions on each local community to its external members to keep community replicas consistent. In subsequent odd super-steps, after processing received vertex action messages, community replicas in different sub-networks

```
ALGORITHM: NotifyVertexCommunities(sub-network)

for (each vertex V in the sub-network) do
    for (each neighbor N of V) do
        if (N is a local vertex) then
            get local vertex N;
            directly set V's communities for N;
        else
            SendMsg(N,[operation=notifyCommunity, V and its communities]);
        end if
    end for
end for
```

Fig. 7. Sending joining communities of vertexes in a sub-network.

```
ALGORITHM: Compute(sub-network)

if (super-step == 0) then
    NotifyVertexNeighbors(sub-network);
else if (super-step == 1) then
    ProcessNeighborMessages(sub-network);
    InitializeCommunities(sub-network);
    NotifyNewCommunities(sub-network);
    NotifyVertexCommunities(sub-network);
else if ((super-step % 2) == 0) and (super-step ≤ MAXSTEP)) then
    ProcessNewCommunityMessages(sub-network);
    ProcessVertexCommunityMessages(sub-network);
    EvolveCommunities(sub-network);
    NotifyVertexActions(sub-network);
else if ((super-step % 2) == 1) and (super-step ≤ MAXSTEP)) then
    ProcessActionMessages(sub-network);
    MergeCommunities(sub-network);
    NotifyNewCommunities(sub-network);
    NotifyVertexCommunities(sub-network);
else
    ProcessActionMessages(sub-network);
    Set all local vertexes to be halt;
end if
```

Fig. 8. The framework of the PSOCD algorithm.

are kept in a consistent state. Then the algorithm merges contained communities if possible. Finally, the algorithm notifies external neighbors of members of the retained communities the local communities, and prepares for the next round evolving.

Theoretically, the algorithm will terminate if there is no vertex left that is changing its communities. However, such a graceful termination may not be achieved due to a few vertices who, in a cyclic way, repeat the same decisions within several sequential evolving generations, i.e., leaving from a community and later joining into it again, because of mutual influence of actions of different vertexes. Therefore, the algorithm sets a maximum evolving generations to make sure it will terminate eventually.

F. Considerations on Implementation

In a swarm intelligence system each individual keeps its state and interacts with its neighbors and the system environment to make its own decisions independently. For the community detection problem the state of a vertex includes its neighbors, neighbors' neighbors, joining communities, external neighbors' communities, and more. Clearly, the communities saved by vertices of a sub-network have a lot of replicas in a working machine node. To keep these replicas being consistent, a message must be sent to each vertex who holds a replica. Note that in Giraph++ system, a sub-graph programming model is supported, such that all local vertexes of a sub-network could share some variables among them. Therefore, in our implementation, communities used locally are held by a sub-network and identified by unique identifiers across the whole system and a vertex only saves IDs of communities into which itself joins or its external neighbors join. There are two advantages in following this approach: (1) the consumed memory is reduced; (2) the messages needed to keep community replicas being consistent are decreased greatly. The reason for that is that if one vertex updates the state of a community, all vertices referencing this community will be aware of the changes immediately as well. Therefore, for those messages related to community state updating(including community notification messages and vertex action notification messages), only one message needs to be sent to a selected representative vertex from a sub-network and the representative vertex is then responsible for updating its local community replica. As a result, such a message needs to be sent for the times of at most the number of sub-networks. In addition, when a vertex notifies its external neighbors of its joining communities, it only sends the IDs of the joining communities which is much smaller in size than the sending of the whole structures of communities.

As the evolution phase progresses, some communities may no longer be referenced by any vertex. To remove these unnecessarily saved communities from the local community structure of a sub-network, at the beginning of each evolving round, the algorithm also examines all communities and deletes null referenced ones.

G. Time Complexity Analysis

It is difficult to analyze the time complexity of a parallel algorithm in distributed running. In this section, we focus on the algorithm execution of one sub-network and analyze the time complexity of the two major operations of the PSOCD algorithm, i.e., the initialization and evolution phases. The used labels are listed in Table I. We assume all vertices are divided approximately equally into a number of sub-networks, and network communication time is not taken into consideration.

The initialization phase consists of two super-step computations. The first (super-step 0) consists of each vertex sending information about its neighbors to all its neighbors. In the second (super-step 1) the sub-network processes received neighbor notification messages and finds a 3-clique initial community for each local vertex. The time complexity of the sending neighbor information is $O(\bar{N}_V \times \bar{N}_N)$, and the worst time complexity of the finding procedure is $O(\bar{N}_V \times \bar{N}_{LN}^2)$ for finding two connected local neighbors. The complexity of message processing is same as message sending since a vertex is an external neighbor of its external neighbors too. As a result, the worst time complexity of initializing is $O(\bar{N}_V \times \bar{N}_N + \bar{N}_V \times \bar{N}_{LN}^2 + \bar{N}_V \times \bar{N}_{EN})$. If $\bar{N}_N$, $\bar{N}_{LN}$ and $\bar{N}_{EN}$ are small numbers comparing with $\bar{N}_V$, the time complexity becomes approximately linear. Fortunately, with a sub-network having good community structure, finding a 3-clique with two local neighbors for most vertexes needs not N_{vLN}^2 times searches, especially for vertexes with a large number of local neighbors. When using a good partition method, the number of cross-sub-network edges is as less as possible, thus the $\bar{N}_{EN}$ will not be too large, either.

The evolution phase is executed in a number of iterations and each iterations consists of two super-steps as well. In the first super-step (even super-step), the sub-network processes received messages of new communities and vertex communities notification, and evolves communities for each local vertexes. Finally, the sub-network sends community updates (vertex joining and leaving) to selected representative vertexes of other sub-networks if a replica exists for the purpose of keeping them being consistent. The worst time complexity of evolving vertex communities is $O(\bar{N}_V \times \bar{N}_N \times \bar{N}_{VC})$ for each vertex checking all candidate communities for joining. The worst time complexity of sending community update messages is $O(\bar{N}_C \times N_P)$.

In the second super-step (odd super-step), the sub-network processes received community update messages, and merges communities from a vertex viewpoint if possible. Then it sends new community messages to external neighbors (but not members of the community) of community members, and community IDs of each vertex to its external neighbors. The worst merging time complexity is $O(\bar{N}_V \times \bar{N}_{VC}^2)$ for checking each community if it is contained in another one. The worst time complexity of sending new communities notification messages is $O(\bar{N}_C \times N_P)$, and the time complexity of sending vertex communities notification messages is $O(\bar{N}_V \times \bar{N}_{EN})$. Note that, as the first time evolving vertex communities, the new communities and vertex communities notification messages are sent at the end of the initialization phase.

Therefore, the worst time complexity of one evolution iteration is $O(\bar{N}_V \times \bar{N}_N \times \bar{N}_{VC} + \bar{N}_V \times \bar{N}_{VC}^2 + 2\bar{N}_C \times N_P + 2\bar{N}_V \times \bar{N}_{EN})$. The $\bar{N}_C$ usually becomes small at later evolution iterations. If $\bar{N}_N$, $\bar{N}_{VC}$, and $\bar{N}_{EN}$ are small numbers comparing with $\bar{N}_V$, the time complex is approximately linear, too.

When whole cluster is considered, the performance of the initialization or one evolution iteration depends on the worst of sub-networks processing by a number of working machine nodes, plus the worst network communication time. Generally, it is a sensible assumption that the network bandwidth is sufficient and thus network communication will not become the bottleneck of the algorithm's execution.

IV. Verification

In this section the performance of our proposed algorithm will be verified and compared against the SLPA [12] and the OSLOM [21], two algorithms known as the best for networks with either low overlapping density or high overlapping density [6].

TABLE I. TIME COMPLEX ANALYSIS LABELS

Varable	Description
N_P	sub-network number
$\bar{N}_V$	averaged number of vertexes in a sub-network
$\bar{N}_C$	averaged number of local communities in a sub-network
$\bar{N}_N$	averaged number of neighbors of a vertex
$\bar{N}_{LN}$	averaged number of local neighbors of a vertex
N_{vLN}	the number of local neighbors of vertex v
$\bar{N}_{EN}$	averaged number of external neighbors of a vertex
$\bar{N}_{VC}$	averaged number of joined communities of a vertex

TABLE II. LFR PARAMETER SETTING

Parameter	Description	Experiment Setting
N	number of vertexes	100,000
k	average degree of vertexes	40
maxk	maximum degree	100
μ	mixing parameter	from 0.1 to 0.7,increased by 0.1.
minc	minimum community size	20
maxc	maximum community size	100
on	number of overlapping vertexes	from 10% to 80% of N, increased by 10 percent.
om	number of memberships of overlapping vertexes	from 1 to 10,increased by 1.
others	other parameters	default values.

A. Network Data

We use the LFR model [40] synthesized complex networks as evaluation data set. The parameter-settings can be found in Table II. 25 networks were generated in total with different parameter combinations. Each network is divided into 5 approximately equal-sized sub-networks by the Metis algorithm.

B. Community Structure Quality Measurement

To evaluate the quality of the detected community structure, we adopt the Normalized Mutual Information (NMI) for overlapping community structure (denoted as ONMI) [41] to measure the similarity between the detected community structure and the real ground truth. In addition, to assess the quality of detected overlapping vertexes, the recall, precision and F-score metrics are employed.

C. Experiment Setup

We deployed the PSOCD algorithm on a cluster with 6 virtual machines on the Amazon Web Service (AWS). Each virtual machine has 4 core CPUs and 16GB memory.

The maximum compution super-steps in experiments is set as 30. During the first half number of evolution iterations, the threshold of connection strength ratio of vertex joining is set as 0.8, in order to get a high quality core community structure, and then in the second half, the threshold is reduced to 0.5 or even less, for the purpose of getting more complete communities.

In the run of the SLPA algorithm, its parameter r takes all possible values and we picked out the best result as the compared one. The three algorithms are run 30 times on each test network.

D. Results Analysis

We computed the average results of 30 runs and their standard deviations. Fig.9 shows the ONMI results for test networks. As seen from the results, (1) while μ changes from 0.1 to 0.7, the ONMIs of the PSOCD algorithm detected community structures are always better than those of the SLPA algorithm and OSLOM algorithm discovered, especially when μ is set as 0.7. (2) As on increases from 10% of total vertex number to 80%, the ONMIs of all three algorithms are decreasing greatly, except as the percentage reaching 70% and 80% for the OSLOM where the ONMI is slightly increaing. However, those of the PSOCD are much better than those of the two others, particularly for high overlapping density networks. (3) While the overlapping memberships om increases from 1 to 10, again the ONMIs of PSOCD are much better than those of the SLPA and the OSLOM except $om = 1$ and 2. The more the gains, the larger the om.

As for the quality of discovered overlapping vertexes, only the F-scores of test networks is given in Fig.10 due to space limit. It confirms that the performance of the PSOCD is better than those of the SLPA and the OSLOM, as a whole. In fact, the precisions of the SLPA and the OSLOM are 100% for a number of tested networks, and are slightly better than those of the PSOCD. However, the margins between the two algorithms are small if the SLPA and the OSLOM are better and the precisions of the PSOCD are fairly good, particularly for high overlapping density networks. Moreover, the PSOCD could find out the overwhelming majority of true overlapping vertexes, and its recalls are always better than those of the two others, especially for overlapping density and high overlapping diversity networks.

The distributions of detected overlapping memberships of one run for networks with om changing are shown in Fig.11(excluding $om = 1$). It is clear that the PSOCD is able to correctly identify most of the overlapping memberships, even at high overlapping diversity, while the SLPA and the OSLOM can only correctly identify small overlapping memberships. As om increases further, the overlapping memberships identified by the SLPA and the OSLOM are dispersed, and the real ones cannot be discerned. Note that we set the overlapping vertex number to be 10% percent of all vertices.

Therefore, it is safe to conclude that the performance of the proposed PSOCD is better than that of the SLPA and the OSLOM, and the PSOCD is an appropriate choice for high overlapping density or high overlapping diversity network analysis.

V. CONCLUSION

In this paper, we proposed a parallel self-organizing community detection algorithm called PSOCD, which is based on the idea of swarm intelligence. The PSOCD algorithm first divides a large scale network into a number of sub-networks and treats each sub-network as a swarm intelligence sub-system. Each vertex of a sub-network, as an individual, can make its own decisions to join in or leave from communities mainly according to a predefined rule, if the ratio of its connection strength with a candidate community to the maximum connection strength exceeds a designated threshold.

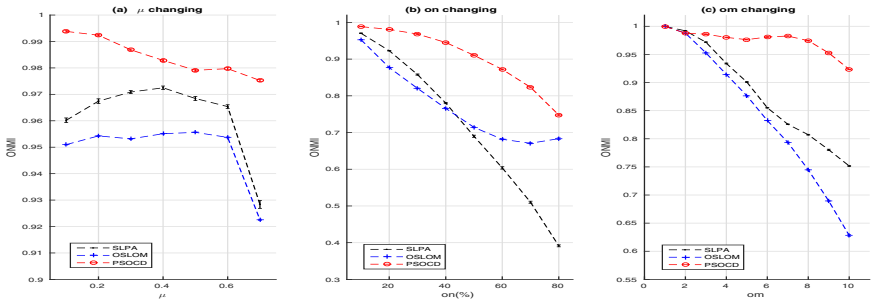

Fig. 9. ONMIs of test synthesized networks.

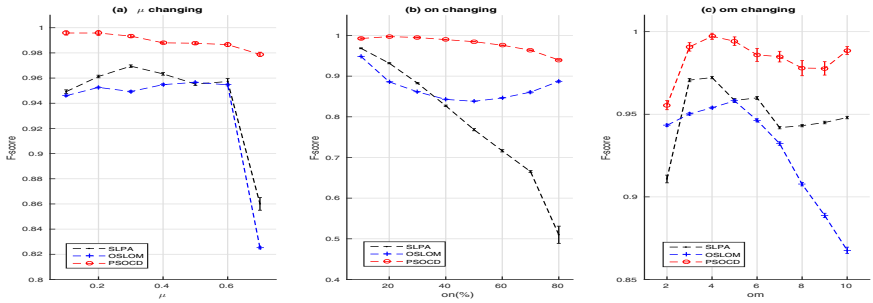

Fig. 10. F-scores of test synthesized networks.

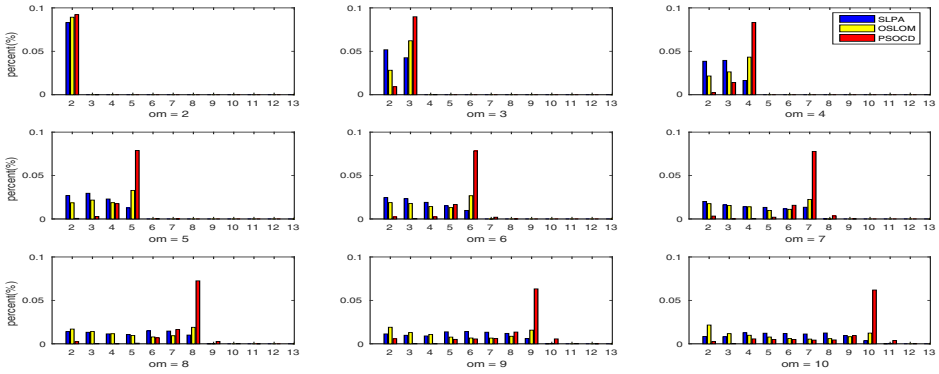

Fig. 11. Distributions of overlapping memberships.

Local communities of each sub-network are sent to other sub-networks if needed to make their members have a chance to join in, and vertex actions of a sub-network are sent as well to guarantee community replicas being in consistent state. By having all vertices iteratively making these decisions over a number of generations, an optimal community structure of the whole large network is emerging gradually. Theoretically, when no vertex acts any more (joins or leaves a community), the algorithm terminates as it is assumed that an optimal community structure is reached. However, a few number of vertices may take evolve a cyclic behavior within several sequential evolution iterations, i.e., repeatedly leaving from a community to later joining it again, due to mutual influence of actions of different vertices. Thus in our implementation we limited the maximum evolving generations to make sure the algorithm terminates.

ACKNOWLEDGMENT

The authors would like to thank Amazon Web Services for providing resources for us to conduct experiments. This work was supported by the Shaanxi Science and Technology Innovation Project plan (No. 2016KTZDGY04-01), Project of Natural Science Foundation Research Project of Shaanxi Province (No. 2016JM6048), Science and Technology Project of Shaanxi Province Science and Technology Department (No. 2016GY-092), Special Research Program of Shaanxi Provincial Department of Education (No. 16JK1687), and New Star Team of Xi'an University of Posts & Telecommunications.

REFERENCES

[1] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi, "Defining and identifying communities in networks," *Proceeding of the*

National Academy of Sciences of the United States of American, vol. 101, no. 9, pp. 2658–2663, 2004.

[2] G. Palla, I. Derenyi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, no. 7043, pp. 814–818, 2005.

[3] M. Coscia, F. Giannotti, and D. Pedreschi, "A classification for community discovery methods in complex networks," *Statistical Analysis and Data Mining*, vol. 4, no. 5, pp. 512–546, 2011.

[4] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3, pp. 75–174, 2010.

[5] A. Amelio and C. Pizzuti, "Overlapping community discovery methods: A survey," in *Lecture Notes in Social Networks*, ule Gndz-dc and A. ima Etaner-Uyar, Eds. Vienna: Springer, 2014, ch. Social Networks: Analysis and Case Studies, pp. 105–125.

[6] J. Xie, S. Kelley, and B. K. Szymanski, "Overlapping community detection in networks: The state-of-the-art and comparative study," *ACM Computing Surveys*, vol. 45, no. 4, p. 43, 2013.

[7] M. Planti and M. Crampes, "Survey on social community detection," in *Computer Communications and Networks*, N. Ramzan, R. van Zwol, J.-S. Lee, K. Clver, and X.-S. Hua, Eds. London: Springer, 2013, ch. Social Media Retrieval, pp. 65–85.

[8] A. Lancichinetti and S. Fortunato, "Community detection algorithms: a comparative analysis," *Physical review E*, vol. 80, no. 5, p. 056117, 2009.

[9] S. Gregory, "Finding overlapping communities in networks by label propagation," *New Journal of Physics*, vol. 12, no. 10, p. 103018, 2010.

[10] Q. Dai, M. Guo, Y. Liu, and L. Chen, "Mlpa: Detecting overlapping communities by multi-label propagation approach," in *Proceedings of the 2011 IEEE 11th International Conference on Data Mining Workshops.*, vol. Proc of 2013 IEEE Congress on Evolutionary Computation, 2013, pp. 681–688.

[11] Z. Wu, Y. Lin, S. Gregory, H. Wan, , and S. Tian, "Balanced multi-label propagation for overlapping community detection in social networks," *Journal of Computer Science and Technology*, vol. 27, no. 3, p. 468479, 2012.

[12] J. Xie, B. K. Szymanski, and X. Liu, "Slpa: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process," in *Proceedings of the 2011 IEEE 11th International Conference on Data Mining Workshops.*, 2011, pp. 344–349.

[13] Z. Liang, F. Y. Jianping Li, and P. Athina, "Detecting community structure using label propagation with consensus weight in complex network," *Chinese Physics B*, vol. 23, no. 9, p. 098902, 2014.

[14] H. Sun, J. Huang, Y. Tian, Q. Song, and H. Liu, "Detecting overlapping communities in networks via dominant label propagation," *Chinese Physics B*, vol. 24, no. 1, p. 018703, 2015.

[15] J. Baumes, M. K. Goldberg, M. S. Krishnamoorthy, and N. Preston, "Finding communities by clustering a graph into overlapping subgraphs," in *Proceedings of the IADIS International Conference on Applied Computing*, 2005, p. 97104.

[16] J. Baumes, M. Goldberg, and M. Magdon-Ismail, "Efficient identification of overlapping communities," in *Proceedings of the 2005 IEEE International Conference on Intelligence and Security Informatics*, 2005, p. 2736.

[17] A. Lancichinetti, S. Fortunato, and J. Kertesz, "Detecting the overlapping and hierarchical community structure of complex networks," *New Journal of Physics*, vol. 11, p. 033015, 2009.

[18] Y. Li and Z. Zhu, "A fast method of detecting overlapping community in network based on lfm," *Journal of Software*, vol. 10, no. 7, pp. 825–834, 2015.

[19] F. Wei, W. Qian, W. Chen, and A. Zhou, "Detecting overlapping community structures in networks," *World Wide Web*, vol. 12, no. 2, p. 235261, 2009.

[20] M. Aaron and H. N. J, "Detecting highly overlapping communities with model-based overlapping seed expansion," in *Proceedings of the 2010 International Conference on Advances in Social Networks Analysis and Mining*, 2010, p. 112119.

[21] A. Lancichinetti, F. Radicchi, J. J. Ramasco, and S. Fortunato, "Finding statistically significant communities in networks," *PLoS ONE*, vol. 6, no. 4, p. e18961, 2011.

[22] M. Coscia, G. Rossetti, F. Giannotti, and D. Pedreschi, "Uncovering hierarchical and overlapping communities with a local-first approach," *Transactions on Knowledge Discovery from Data*, vol. 9, no. 1, p. 6, 2014.

[23] E. H. W. Xu and P. M. Hui. (2015, Sep) Efficient detection of communities with significant overlaps in networks: Partial community merger algorithm. [Online]. Available: http://arxiv.org/pdf/1509.00556v1.pdf

[24] B. S. Rees and K. B. Gallagher, "Overlapping community detection by collective friendship group inference," in *Proc of 2010 International Conference on Advances in Social Networks Analysis and Mining*, 2010, pp. 375–379.

[25] R. Shang, S. Luo, Y. Li, L. Jiao, and R. Stolkin, "Large-scale community detection based on node membership grade and sub-communities integration," *Physica A: Statistical Mechanics and its Applications*, vol. 428, pp. 279–294, 2015.

[26] W. Chen, Z. Liu, X. Sun, and Y. Wang, "A game-theoretic framework to identify overlapping communities in social networks," *Data Mining and Knowledge Discovery*, vol. 21, no. 2, p. 224240, 2010.

[27] H. Alvari, S. Hashemi, and A. Hamzeh, "Discovering overlapping communities in social networks: A novel game-theoretic approach," *AI Communications*, vol. 26, no. 2, pp. 161–177, 2013.

[28] ——, "Detecting overlapping communities in social networks by game theory and structural equivalence concept," in *Proceedings of the Third international conference on Artificial intelligence and computational intelligence*, September 2011, pp. 620–630.

[29] A. Hajibagheri, H. Alvari, A. Hamzeh, and S. Hashemi, "Social network community detection using the shapley value," in *2012 16th CSI International Symposium on Artificial Intelligence and Signal Processing*, May 2012, pp. 222–228.

[30] C. Pizzuti, "Overlapped community detection in complex networks," in *Proc of the 11th Annual conference on Genetic and Evolutionary computation*, 2009, p. 859866.

[31] X. Zhou, Y. Liu, J. Zhang, T. Liu, and D. Zhang, "An ant colony based algorithm for overlapping community detection in complex networks," *Physica A: Statistical Mechanics and its Applications*, vol. 427, pp. 289–301, 2015.

[32] B. S. Rees and K. B. Gallagher, "Overlapping community detection using a community optimized graph swarm," *Social Network Analysis and Mining*, vol. 2, no. 4, pp. 405–417, 2012.

[33] ——, "Detecting overlapping communities in complex networks using swarm intelligence for multi-threaded label propagation," in *Studies in Computational Intelligence*. Berlin Heidelberg: Springer, 2013, ch. Complex Networks, pp. 111–119.

[34] T. Chakraborty, "Leveraging disjoint communities for detecting overlapping community structure," *Journal of Statistical Mechanics*, vol. 2015, no. 5, p. 05017, 2015.

[35] X. Wang, L. Jiao, and J. Wu, "Adjusting from disjoint to overlapping community detection of complex networks," *Physica A*, vol. 388, 2009.

[36] T. Chakraborty, N. Park, and V. Subrahmanian, "Ensemble-based algorithms to detect disjoint and overlaping communities in networks," in *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis (ASONAM),*, San Franciso, August 2016.

[37] Y. Tian, A. Balmin, S. A. Corsten, S. Tatikonda, and J. McPherson, "From 'think like a vertex' to 'think like a graph'," in *Proceedings of the VLDB Endowment*, vol. 7, no. 13, November 2013, pp. 193–204.

[38] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, p. 359392, 1998.

[39] T. Chakraborty, S. Sriram, and G. Niloy, "On the permanence of vertices in network communities," in *Proc of the 20th ACM SIGKDD international conference on Knowledge discovery and data miningt*, 2014, pp. 1396–1405.

[40] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Physical Review E*, vol. 78, no. 4, p. 046110, 2008.

[41] A. F. McDaid, D. Greene, and N. Hurley. (2013, Aug) Normalized mutual information to evaluate overlapping community finding algorithms. [Online]. Available: https://arxiv.org/pdf/1110.2515v2.pdf