Advance Access publication on 26 September 2014
doi:10.1093/comjnl/bxu092

Finding a Friendly Community in Social Networks Considering Bad Relationships

Sang-Yeon Kim, Dong-Wan Choi and Chin-Wan Chung*

Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), 291 Daehak-ro, Yuseong-gu, Daejeon, Republic of Korea
**Corresponding author: chungcw@kaist.edu*

Community detection in social networks is one of the most active problems with lots of applications. Most of the existing works on the problem have focused on detecting the community considering only the closeness between community members. In the real world, however, it is also important to consider bad relationships between members. In this paper, we propose a new variant of the community detection problem, called _friendly community search_. In the proposed problem, for a given graph, we aim to not only find a densely connected subgraph that contains a given set of query nodes but also minimizes the number of nodes involved in bad relationships in the subgraph. We prove that is Non-deterministic Polynomial-time hard (NP-hard), and develop two novel algorithms, called Greedy and SteinerSwap that return the near optimal solutions. Experimental results show that two proposed algorithms outperform the algorithm adapted from an existing algorithm for the optimal quasi-clique problem.

Keywords: community detection; social network; graph algorithm

Received 15 November 2013; revised 29 June 2014
Handling editor: Jong Hyuk Park

1. INTRODUCTION

With the emergence of various large social network services such as Facebook,[1] Twitter[2] and Google+,[3] *community detection* in social networks is getting a great deal of attention in a variety of contexts. In many applications based on social networks, it is required to find a meaningful community structure that can reveal some interesting characteristics behind the target social network [1]. Recently, there were also some attempts to find a community including a given set of query nodes, which is called the *community search* problem [2] distinguished from the classic community detection problem. This query-dependent variant of the community detection problem can be used in many applications such as finding thematic groups and organizing social events. For example, the host of a social event may want to invite people who are close to the host and well acquainted with each other. If the members of the social community are close to each other, the chance of success of the social event will become higher [2]. This research addresses that the meaningful community should be a subgraph with a high density and a small diameter.

However, only considering these properties is not always sufficient to find a meaningful community consisting of members truly acquainted with each other. In the real world, people build many relationships with other people for various reasons such as the same work place, the same school or the same neighbourhood. In these relationships, there can be some bad relationships as well as good relationships. For instance, people may be unwilling to share their private activities with other members who are connected in the social network, or they can even dislike their acquaintances due to some personal reasons. These relationships are regarded as bad relationships.

For the truly *friendly* community, bad relationships are better to be avoided. According to a research result in neuroscience it is known that being together with someone whom people dislike can adversely affect on the activity of their brain [3]. This implies that people are not likely to enjoy social events

[1]https://www.facebook.com/.
[2]https://twitter.com/.
[3]https://plus.google.com/.

or achieve the desired teamwork, when they belong to a community with someone in a bad relationship. If there are many members who are involved in bad relationships in the community, the community cannot be very friendly, which leads to the fail of the original purpose of the community. In order to improve the chance of achieving the goal of the community, it is important to minimize the number of members involved in bad relationships in the community as well as maximize the closeness of the community.

However, most existing studies have focused on finding a community considering only the closeness in social networks, and hence the resulting community is mainly a tightly coupled subgraph. Unfortunately, the straightforward adaptation of these works on the community detection problem is not effective to find a *friendly community* in which the number of members involved in bad relationships is minimized. For instance, the goal of the community search problem is finding a community including a set of given query nodes while trying to maximize the density of the community. However, the density of the community may not capture bad relationships between members, since the density of the community only represents how the members of the community are closely connected in social networks.

Motivated from this limitation, we propose a new problem that is an important variant of the community detection problem, called the *friendly community search* (FCS) problem. For a given graph and a set of query nodes in the graph, we aim to find a densely connected subgraph that contains the set of query nodes while minimizing the number of nodes (i.e. members) involved in bad relationships in the subgraph. The FCS problem can also be useful in any applications where the community search problem is utilized.

The first goal of the FCS problem is to find a densely connected subgraph that contains query nodes. Therefore, we need to choose an appropriate density measure. There are three well-known measures that represent the density of a graph: the edge density, the average degree and the minimum degree of the nodes are commonly used. Among these measures, we use the edge density, which is a ratio of the number of edges to the number of all possible pairs of nodes. This is due to our observation that the actual density of a graph is captured better by the edge density than the other measures, which will be further explained in Section 3.

Our second goal is to minimize the number of members involved in bad relationships in the resulting subgraph. We denote the bad relationships between nodes by *hate set* H, and we define the *hate function* f to measure the bad relationships in the subgraph based on H. The smaller f value of the subgraph, the less members involved in bad relationships in the subgraph.

It is challenging to find the community satisfying both of these two criteria. We theoretically prove this by showing that the FCS problem is Non-deterministic Polynomial-time hard (NP-hard). To solve the FCS problem, we devise two effective algorithms returning a nearly optimal solution. The first is a greedy algorithm, called GREEDY, where the nodes that minimize f are iteratively removed from the entire graph. The second algorithm, called STEINERSWAP, is based on the idea opposite to GREEDY. Instead of removing nodes, STEINERSWAP adds nodes to the seed structure which is the Steiner tree built on a given set of query nodes. By doing this, we can achieve a result similarly to that of GREEDY in much less execution time.

The contributions of this paper are as follows:

(i) To the best of the our knowledge, we are the first to consider bad relationships between the members of a community, and thereby proposing a novel problem of finding the densely connected community that minimizes the number of members involved in bad relationships in the community, namely the *FCS* problem. Also, we prove that the proposed problem is NP-hard.
(ii) We propose two effective algorithms, namely GREEDY and STEINERSWAP. The GREEDY algorithm iteratively removes nodes from the entire graph while the STEINERSWAP algorithm adds nodes to the Steiner tree built on a set of query nodes to find a community with low time complexity.
(iii) We experimentally evaluate the effectiveness of two proposed algorithms by using real datasets. Experimental results show that two proposed algorithms are more effective than an adaptation of an existing algorithm.

The rest of the paper is organized as follows. In Section 2, we survey previous works relevant to the FCS problem. The formal definition of the FCS problem and its NP-Hardness are presented in Section 3. The details of two proposed algorithms for the FCS problem are explained in Section 4. In Section 5, we experimentally evaluate the effectiveness of the proposed algorithms. Finally, we conclude the paper in Section 6.

2. RELATED WORK

Our paper is inspired by existing works on detecting communities and those on finding a dense subgraph. In this section, we survey both lines of works as follows.

Community detection. The problem of detecting a community in a large graph is important in various disciplines, such as sociology, biology and computer science [4]. Girvan and Newman [1] introduced the *modularity* measure to find a community. There are many papers working on the modularity measure and developing algorithms based on the modularity [5–7]. Moreover, Martelot and Hankin [8] proposed a algorithm to detect community on large-scale networks. So far, the aforementioned works can be classified as the community detection on unsigned graphs in that the weight of each edge cannot be negative. There are another type of works on signed networks where each edge can have either a positive weight or a negative weight [9–11]. Basically, most of the existing works on community detection do not consider any query nodes, which differentiate our problem from the community detection problem. In addition, we use the edge-density measure to find an appropriate community instead of other measure such as the modularity.

Community search. Recently, Sozio and Gionis [2] proposed the community search problem, which is a query-dependent variant of the community detection problem. Given a graph, a set of query nodes, the problem aims to find a densely connected subgraph, which contains the set of query nodes. The minimum degree of nodes in the subgraph is used to measure the density of the subgraph. There are two main differences of the FCS problem from the community search problem. First, we use the edge density to measure the density of the subgraph since the minimum degree of nodes may not capture the exact density of a subgraph. Secondly, we consider not only the density of the subgraph but also bad relationships in the subgraph.

Pairwise constrained clustering. The pairwise constrained clustering problem is one of the variants of the community detection problem, where objects and pairwise relationships for each object are given. The pairwise relationships are classified into as two types, namely *must-link* and *cannot-link*. If two objects are in the must-link, they should be in the same community. However, if two objects are in the cannot-link, they should be split into different communities. The pairwise constrained clustering problem is finding communities which satisfy all pairwise constraints yet minimize the objective function. The first work of the pairwise constrained clustering problem is proposed by Wagstaff and Cardie [12], and there are several enhanced works later [13–15]. The FCS problem is different from the pairwise constrained clustering problem in two folds. First, the FCS problem is the community detection problem based on the graph characteristics such as the density, but the pairwise constrained clustering problem is not. Secondly, the bad relationship between members in the community is different from the cannot-link between objects, since the resulting community of the FCS problem can include members who are in bad relationships.

Team formation. Lappas *et al.* [16] studied another type of problems of detecting a community in social networks, called *team formation*. Given a social network graph where each individual node in the graph is associated with a set of skills, and a given task that requires a set of skills, the problem aims at finding a subgraph that covers the given set of skills, that is, a team to perform the task. There are many works and algorithms on the team formation problem [17–20]. In the team formation problem, the communication cost is used to measure the effectiveness of a solution. However, a bad relationship between two nodes cannot be represented as a communication cost between two nodes, since the communication cost is usually based on the shortest distance between two nodes. For example, when the bad relationship between two nodes A and B is represented as an edge with a large weight, the communication cost between A and B can be determined by the shortest distance that is smaller than the edge weight between A and B.

Densest subgraph. There are several works [21, 22] that find a densest subgraph in a given graph. Let $G = (V, E)$ be an undirected graph. The goal of the *densest-subgraph* problem is to find a set $S \subseteq V$ that maximizes the average degree of nodes in S. The densest-subgraph problem can be solved optimally in linear time by using the parametric maximum flow algorithm [21]. Charikar [22] introduced a 2-approximation greedy algorithm for finding a dense subgraph in a given graph. However, the average degree of nodes can lead to an inappropriate resulting subgraph, called the *free-rider* effect [2]. In the FCS problem, we use the edge density to find a densely connected subgraph.

Quasi-clique. Another approach finding a dense subgraph is based on the *quasi-clique*. Let $G = (V, E)$ be an undirected graph. Then a set of nodes is called a γ-quasi-clique or a γ-clique for $\gamma \in [0, 1]$, if the edge density of the induced subgraph by the set of nodes is greater than or equal to γ. Uno [23] proposed a polynomial delay algorithm for finding all γ-quasi-cliques in a graph for given γ. Recently, Tsourakakis *et al.* [24] introduced a general framework for finding an optimal quasi-clique based on the concept of the *edge surplus*. To solve the problem, two algorithms are proposed: a greedy algorithm with an additive approximation and a heuristic algorithm based on the local-search paradigm. Moreover, the *Constrained-OQC* problem [24], which asks an optimal quasi-clique containing query nodes, is introduced and proved to be NP-hard. The main difference of our problem from the Constrained-OQC problem is that our resulting community should be connected, which is not always the case in the Constrained-OQC problem.

3. FCS PROBLEM

In this section, we formally define the FCS problem in Section 3.1, and prove the NP-hardness of the FCS problem using the Steiner tree problem in Section 3.2

3.1. Problem definition

In this section, we introduce our notation, and formally define the FCS problem.

Let us assume that we have an unweighted undirected social network $G = (V, E)$, where V is the set of nodes (users), and E is the set of edges (relationships). We denote by n the number of nodes and by m the number of edges. The edge between two nodes $u, v \in V$ implies that u and v are connected in the social network. For a set of nodes $S \subseteq V$, let $G[S] = (S, E[S])$ be the subgraph induced by S, $E[S]$ be the set of edges both of whose end nodes are in S.

For each node in the graph, we define the bad relationships of nodes by using a set of nodes, called the hate set H, which is formally defined as follows.

Definition 3.1 (Hate set). *Let $G = (V, E)$ be the unweighted undirected graph. For any node $v \in V$, a hate set, denoted by H_v, is a set of nodes that are hated by v such that $\forall u \in H_v, \exists (u, v) \in E$.*

We also use $\mathcal{H} = \{H_v | v \in V \wedge H_v \neq \emptyset\}$ to denote the collection of all the hate sets of the nodes. Based on $\mathcal{H}$, we define the following hate function f that measures the number of nodes that are involved in any bad relationships.

Definition 3.2 (Hate function). *Let $G = (V, E)$ be the unweighted undirected graph, $S \subseteq V$ be a subset of the nodes and H_v be the hate set of the node $v \in V$. Then the hate function $f(S)$ that assigns an integer value to $S \in 2^V$, i.e. $f : 2^V \to \mathbb{Z}$, is defined as follows*:

$$f(S) = |\{v \mid v \in S \wedge \exists u \in (H_v \cap S)\}|.$$

We are setting the intuitive hate function as the number of nodes involved in bad relationships. There can be other candidate measures used to define the hate function, such as the number of bad relationships of a community, and the relative quantity of the number of members in bad relationships in a community. In case of the number of bad relationships involved in the community, we cannot distinguish different cases, thus cannot select a better community. For example, suppose two cases: (1) a member hates all member in the community and (2) all members hate a member in the community. Even though these two cases should be differently measured, the numbers of bad relationships are the same in both cases. Also, when we use the relative quantity of the number of members in bad relationships, there can be too many tied solutions which minimize the objective function. In addition, from our experiment, our algorithms are shown to be effective for our measure as well as this measure.

Based on the hate function f, a subgraph, the node set of which has a smaller f value, is regarded as a *more friendly* community in the sense that, in such a community, there will be only a small number of members who hate or dislike other members in the community.

As mentioned earlier, we need to choose an appropriate measure that captures the density of a graph. The average degree has the free-rider effect that nodes irrelevant to query nodes can also be attached to the graph since they may increase the total average degree [2]. This is illustrated in Fig. 1a. In this example, the set of query nodes $Q = \{a, d, c\}$ and graph G are given. We know the community $S_1 = \{a, b, c, d, e\}$ as the intuitive solution which includes Q. However, the community that maximizes the average degree is $S_2 = \{a, b, c, d, e, f, g, h, i, j, k\}$. Intuitively, the nodes f, g, h, i

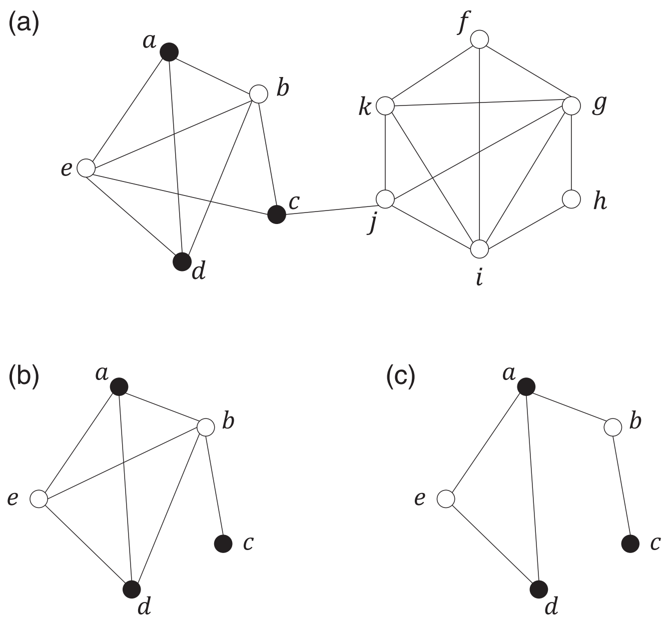

FIGURE 1. Different definitions of the density. (**a**) Graph G, (**b**) subgraph induced by S_3 and (**c**) subgraph induced by S_4.

and k are irrelevant to the set of query nodes Q. Moreover, the minimum degree cannot exactly capture the density of the graph. To illustrate, the set of query nodes $Q = \{a, d, c\}$, and two subgraphs $G[S_3]$ and $G[S_4]$ are shown in Fig. 1b and c, respectively. Intuitively, we can note that $G[S_3]$ is denser than $G[S_4]$. However, the minimum degrees of S_3 and S_4 are the same, which is 1 at node c. Therefore, we decide to use the ratio of the number of edges to the number of all possible pairs of nodes, which is called the edge density $|E[S_3]|/\binom{|S_3|}{2}$, to measure the density of a subgraph $G[S_3]$. Also, as mentioned in Section 2, a subgraph whose edge density exceeds γ is referred to as a γ-clique.

Now, we present the formal definition of the FCS problem as follows.

Definition 3.3 (FCS problem). *Given an unweighted undirected graph $G = (V, E)$, a set of query nodes $Q \subseteq V$, a collection of all hate sets $\mathcal{H}$, the hate function f and a density threshold $\gamma \in [0, 1]$, find a set of nodes $S \subseteq V$ which satisfies the following conditions*:

(1) *S contains $Q (Q \subseteq S)$,*
(2) *$G[S]$ is a connected γ-clique and*
(3) *$f(S)$ is minimized.*

We also refer to any subset $S' \subseteq V$ satisfying only (1) and (2) conditions as a feasible solution for the FCS problem.

3.2. Proof of NP-hardness

The FCS problem is an optimization problem where the objective is to find a subgraph that minimizes the f value. In this section, we prove that our FCS problem is NP-hard.

We first consider the decision version of the FCS problem as the following definition.

Definition 3.4 (Decision version of the FCS problem). *Given an unweighted undirected graph $G = (V, E)$, a set of query nodes $Q \subseteq V$, a collection of all hate sets $\mathcal{H}$, the hate function f, an integer h and a density threshold $\gamma \in [0, 1]$, is there a set of nodes $S \subseteq V$ which satisfies conditions*:

(1) $Q \subseteq S$,
(2) *$G[S]$ is a connected γ-clique and*
(3) $f(S) \leq h$.

It is easy to note that the only difference from Problem 3.3 is condition (3).

Lemma 3.1. *The decision version of the FCS problem is in NP.*

Proof. When a set of nodes $S \subseteq V$ is given, we can obviously verify in a polynomial time whether $f(S)$ is less than or equal to h and $G[S]$ is a connected γ-clique. Therefore, the decision version of the FCS problem is in NP. □

Now, we prove that the decision version of the FCS problem is NP-complete by reducing the decision version of the Steiner tree problem. The Steiner tree problem is a well-known NP-hard problem as proved by Karp [25], and its decision version is defined as follows.

Definition 3.5 (Decision version of the Steiner tree problem). *Given an unweighted undirected graph $G = (V, E)$, a set of query nodes $Q \subseteq V$ and an integer parameter k, is there a subtree of G containing all nodes in Q and having at most k edges.*

In the following lemma, we prove that the decision version of the FCS problem is reducible to the decision version of the Steiner tree problem.

Lemma 3.2. *The decision version of the Steiner tree problem is polynomial-time reducible to the decision version of the FCS problem.*

Proof. The steps for transforming the decision version of the Steiner tree problem defined in Problem 3.5 to the corresponding FCS problem are as follows:

(1) set $G = (V, E)$ as the input graph of the FCS problem,
(2) set Q as the query nodes of the FCS problem,
(3) for any node $v \in V$, set $H_v = \{u|(u, v) \in E\}$,
(4) set $h = k + 1$ and
(5) set $\gamma = 0$.

We show that there is a solution for the Steiner tree problem if and only if there is a solution for the corresponding FCS problem.

Let us consider the Steiner tree problem with G, Q, k, and let T be the set of nodes in the solution Steiner tree. Then, in the FCS problem, $G[T]$ is a connected 0-clique and $f(T) \leq k+1$. Since, for any edges $(u, v) \in E[T]$, u and v are in a bad relationship.

Conversely, the solution for the transformed FCS problem is that for the Steiner tree problem. For the solution of the FCS problem, that is, a set S of nodes, the minimum spanning tree of $G[S]$ is a Steiner tree built on Q with at most k edges, since $|S| \leq h$. □

Theorem 3.1. *The FCS problem is NP-hard.*

Proof. By Lemmas 3.1 and 3.2, the decision problem of the FCS problem is proved to be NP-complete. If an optimization problem has an NP-complete decision version, then the optimization problem is NP-hard [26]. Therefore, the FCS problem is NP-hard □

4. ALGORITHMS FOR THE FCS PROBLEM

Since the FCS problem is NP-hard, it is expensive to find a optimal solution. Therefore, in this section, we devise two algorithms that return solutions which are close to the optimal one in a reasonable running time.

4.1. Greedy algorithm

In this section, we propose the Greedy algorithm for the FCS problem. The Greedy algorithm is an adaptation of the greedy algorithm finding a densest subgraph, which is proposed by Charikar [22].

Algorithm 1: Greedy

```
Input: $G := (V, E)$, $\mathcal{H}$ := a collection of hate sets, $Q$ := a given
       set of query nodes, $\gamma$ := a given density threshold
Output: $S$ := a set of nodes $S \subseteq V$
1 $S_0 \leftarrow$ GreedyQuasi($G, Q, \gamma$)
2 $S \leftarrow$ GreedyRemove($S_0, \mathcal{H}, Q, \gamma$)
3 return $S$
```

As shown in Algorithm 1, Greedy consists of two steps, namely GreedyQuasi and GreedyRemove. In the first step, regardless of the f value, GreedyQuasi focuses on finding a set of nodes $S_0 \in V$ which satisfies the conditions (1) $Q \subseteq S_0$, and (2) $G[S_0]$ is a connected γ-clique. In the second step, GreedyRemove removes nodes from S_0 to minimize $f(S_0)$. In such a way, the final set of nodes $S \subseteq S_0$ is reported.

The pseudocode of the first step algorithm is given in Algorithm 2. Let D denote a set of nodes that cannot be

Algorithm 2: GREEDYQUASI

```
Input: $G := (V, E)$, $Q :=$ a given set of query nodes, $\gamma :=$ a
       given density threshold
Output: $S_0 :=$ a set of nodes $S_0 \subseteq V$
1  $S_0 \leftarrow V, D \leftarrow \emptyset$
2  while $G[S_0]$ is not a $\gamma$-clique and $(S_0 \setminus D) \neq \emptyset$ do
3      $v \leftarrow$ the node $u$ in $(S_0 \setminus D)$ with the smallest degree in $G[S_0]$
4      if $v \notin Q$ and $G[S_0 \setminus \{v\}]$ is connected then
5          $S_0 \leftarrow S_0 \setminus \{v\}$
6      else
7          $D \leftarrow D \cup \{v\}$
8  return $S_0$
```

Algorithm 3: GREEDYREMOVE

```
Input: $S_0$, $\mathcal{H} :=$ a collection of hate sets, $Q :=$ a given set of
       query nodes, $\gamma :=$ a given density threshold
Output: $S :=$ a set of nodes $S \subseteq V$
1  $S \leftarrow S_0$
2  Repeat
3      $R \leftarrow \emptyset$
4      for each $v \in S$ do
5          if $v \notin Q$ and $f(S \setminus R) \geq f(S \setminus \{v\})$ and $G[S \setminus \{v\}]$ is a
           connected $\gamma$-clique then
6              $R \leftarrow \{v\}$
7          if $H_v \cap Q = \emptyset$ and $f(S \setminus R) \geq f(S \setminus H_v)$ and
           $G[S \setminus H_v]$ is a connected $\gamma$-clique then
8              $R \leftarrow H_v$
9      $S \leftarrow S \setminus R$
10 Until $R \neq \emptyset$
11 return $S$
```

removed. GREEDYQUASI finds a node $v \in (S_0 \setminus D)$ with the smallest degree in $G[S_0]$ (Line 3). If v is not in Q and $G[S_0 \setminus \{v\}]$ is connected, then v is removed from S_0 (Lines 4 and 5). Otherwise, v is added to D (Lines 6 and 7). GREEDYQUASI is terminated if $G[S_0]$ is a γ-quasi-clique or $(S_0 \setminus D) = \emptyset$. GREEDYQUASI returns the set of nodes $S_0 \subseteq V$ such that $G[S_0]$ is a connected γ-clique, $Q \subseteq S_0$ for a given γ.

In the second step, the goal of GREEDYREMOVE is minimizing the f value. The pseudocode of GREEDYREMOVE is shown in Algorithm 3. S is initially set to be the returned set of nodes from GREEDYQUASI (Line 1). GREEDYREMOVE first finds the set R of nodes such that $f(S \setminus R)$ is minimized, $G[S \setminus R]$ is a connected γ-clique (Lines 4–8). For $v \in S$, R is set to either $\{v\}$ or H_v to decrease the f value. GREEDYREMOVE iteratively removes R from S at each iteration (Line 9). It is repeated until no nodes can be removed.

The running time of GREEDY is the summation of the running times of GREEDYQUASI and GREEDYREMOVE. The execution time required for GREEDYQUASI is $O(n(n+m)+m)$. This is because we can find the node with minimum degree by using a list of nodes that is indexed by degrees in a constant time, we can check connections between the query nodes in $O(n+m)$, and the total amount of updates in the list of nodes due to the removed node is $O(m)$. The time complexity for GREEDYREMOVE is $O(n(n+m))$, since we need to check the connections for each iteration. Therefore, the time complexity of GREEDY is $O(n(n+m)+m)$.

The step in the GREEDY algorithm with the highest cost is the process of checking connections between query nodes in Q. To reduce this cost, we use the Steiner tree built on the query nodes. If a node is not in the Steiner tree, the node does not affect the connections between query nodes in Q. Therefore, if a node is not in the Steiner tree, GREEDY does not need to check the connections. This technique can reduce the cost of the connection test.

The problem of finding an quasi-clique in a given graph that contains query nodes, is NP-hard as mentioned in Section 2. Thus, the GREEDY algorithm guarantees finding a feasible solution for the given density threshold γ value as the following theorem.

THEOREM 4.1. *Let S^* be the set of nodes of a connected γ-clique which contains query nodes Q. Let us consider a specific iteration I of* GREEDYQUASI *where the node $v \in S^*$ is removed by* GREEDYQUASI. *Let S_I be the set of nodes in that iteration I. The* GREEDY *algorithm guarantees that it is possible to find a feasible solution when $\gamma \leq (|E[S_I]| - (|S_I| - |S^*|))/\binom{|S^*|}{2}$.*

Proof. See Appendix. □

The GREEDY algorithm guarantees to find a feasible solution for the specific γ range. The main disadvantage of GREEDY is due to the fact that it removes nodes from the entire graph. This can lead to a high time complexity.

4.2. SteinerSwap algorithm

The GREEDY algorithm removes nodes from the entire graph and thus it has a high time complexity. In order to reduce the time complexity, we present another heuristic algorithm, called STEINERSWAP. Before we explain the STEINERSWAP algorithm, we introduce a straightforward solution adapted from the existing algorithm for the quasi-clique problem.

A simple adaptation of the LocalSearchOQC [24] algorithm, called BASELINE, can be a solution of the FCS problem. Since query nodes should be connected in the subgraph induced by the resulting node set, BASELINE uses the Steiner tree built on a given set of query nodes as an initial seed set S_0. Then, BASELINE keeps adding nodes to S_0 while the edge density of $G[S_0]$ is improved. When no nodes can be added, BASELINE removes nodes from S_0 which increases the edge density of $G[S_0]$. When no nodes can be removed, BASELINE restarts from the adding step. BASELINE is terminated when $G[S_0]$ is

Algorithm 4: SteinerSwap

```
Input: $G := (V, E)$, $\mathcal{H} :=$ a collection of hate sets, $Q :=$ a given set of query nodes, $\gamma :=$ a given density threshold
Output: $S :=$ a set of nodes $S \subseteq V$
1 $T \leftarrow$ SteinerTree($G, Q$) //Compute Steiner tree
2 $S_0 \leftarrow$ SteinerQuasi($G, T, Q, \gamma$) //Find quasi-clique
3 $S \leftarrow$ HateSwap($G, S_0, \mathcal{H}, Q, \gamma$) //Swap or Remove nodes
4 return $S$
```

Algorithm 5: SteinerTree

```
Input: $G := (V, E)$, $Q :=$ a given set of query nodes
Output: $T :=$ a set of nodes $T \subseteq V$
1 $T \leftarrow v$, where $v$ is a random node from $Q$
2 while $(Q \setminus T) \neq \emptyset$ do
3     Find $u \in (V \setminus T)$ with the minimum distance to $T$
4     if $Path(u, T) \neq \emptyset$ then
5         $T \leftarrow T \cup \{Path(u, T)\}$
6 return $T$;
```

a γ-clique for given γ. However, Baseline can be ineffective because it adds or removes nodes from the seed set, regardless of the f value. Moreover, the effectiveness of Baseline partly depends on the seed set (i.e. the Steiner tree built on query nodes). As either the size of the Steiner tree or the diameter of the Steiner tree increases, the number of edges needed to improve the edge density also increases. This leads to the increase of the number of nodes to be added. Therefore, the larger number of nodes to be added, the higher probability of increasing the f value.

To overcome the limitations of the straightforward solution, we devise a new algorithm based on the Baseline algorithm, namely SteinerSwap, which is shown in Algorithm 4. SteinerSwap proceeds in three steps: SteinerTree, SteinerQuasi and HateSwap. In the first step, SteinerTree computes a Steiner tree built on query nodes. In the second step, SteinerQuasi finds a set of nodes whose induced subgraph is a connected quasi-clique based on the Steiner tree. Finally, HateSwap swaps (or removes) nodes in the returned set of nodes from SteinerQuasi according to their types.

Let us now explain the detail of the SteinerSwap algorithm. In the first step, we use a 2-approximation algorithm, which is proposed by Kou *et al.* [27], to get nodes of the Steiner tree, called SteinerTree. As shown in Algorithm 5, SteinerTree incrementally adds nodes to T. First, SteinerTree finds a node $u \in V$ which has the minimum distance to T (Line 3). If u exists, all the nodes in the shortest path from u to T are added to T (Lines 4 and 5). This process is terminated if all the nodes in Q are in T. Finally, SteinerTree reports T as a set of nodes of the Steiner tree built on query nodes Q.

In the second step, SteinerQuasi finds a set of nodes of a γ-clique as follows. Let S_0 be the nodes of the Steiner tree reported by SteinerTree. As shown in Algorithm 6,

Algorithm 6: SteinerQuasi

```
Input: $G := (V, E)$, $T :=$ a set of nodes, $Q :=$ a given set of query nodes, $\gamma :=$ a given density threshold
Output: $S_0 :=$ a set of nodes $S_0 \subseteq V$
1 $iteration \leftarrow 0$, $S_0 \leftarrow T$
2 while $iteration < I_{max}$ do
3     while $\exists v \in V \setminus S_0$ such that $\frac{|E[S_0]|}{\binom{|S_0|}{2}} \leq \frac{|E[S_0 \cup \{v\}]|}{\binom{|S_0|+1}{2}}$ do
4         $S_0 \leftarrow S_0 \cup \{v\}$
5     while $\exists v \in V \setminus S_0$ such that $\frac{|E[S_0]|}{\binom{|S_0|}{2}} \leq \frac{|E[S_0 \setminus \{v\}]|}{\binom{|S_0|-1}{2}}$ and $G[S_0 \setminus \{v\}]$ is connected do
6         $S_0 \leftarrow S_0 \setminus \{v\}$
7     $iteration \leftarrow iteration + 1$
8     if $G[S_0]$ is $\gamma$-quasi-clique then
9         break
10 return $S_0$
```

Algorithm 7: HateSwap

```
Input: $G := (V, E)$, $S_0 :=$ a set of nodes, $\mathcal{H} :=$ a collection of hate sets, $Q :=$ a given set of query nodes, $\gamma :=$ a given density threshold
Output: $S :=$ a set of nodes $S \subseteq V$
1 $iteration \leftarrow 0$, $S \leftarrow S_0$
2 while $iteration < I_{max}$ do
3     $S_i =$ a set of type-i nodes in $S$
4     while $\exists u, v$ such that $u \in S_2$, $v \in V \setminus S$, $G[(S \setminus \{u\}) \cup \{v\}]$ is connected $\gamma$-clique, and $f((S \setminus \{u\}) \cup \{v\})$ is minimum do
5         $S \leftarrow (S \setminus \{u\}) \cup \{v\}$ //swap type-2
6     while $\exists u, v$ such that $u \in S_3$, $v \in V \setminus S$, $G[(S \setminus \{u\}) \cup \{v\}]$ is $\gamma$-clique, and $f((S \setminus \{u\}) \cup \{v\})$ is minimum do
7         $S \leftarrow (S \setminus \{u\}) \cup \{v\}$ //swap type-3
8     $S \leftarrow GreedyRemove(S, H, Q, \gamma)$ //remove type-4
9     while $\exists v$ such that $v \in V \setminus S$, $G[S \cup \{v\}]$ is $\gamma$-clique, and $H_v \cap S = \emptyset$ do
10        $S \leftarrow S \cup \{v\}$
11    $iteration \leftarrow iteration + 1$
12 return $S$
```

SteinerQuasi keeps adding nodes to the current set S_0 while the edge density of $G[S_0]$ increases (Lines 3 and 4). When no nodes can be added, SteinerQuasi keeps removing nodes which increase the edge density of $G[S_0]$ (Lines 5 and 6). This process continues until the number of iteration exceeds the maximum number of iteration $I_{\max}$ (i.e. 50).

In HateSwap, let S_0 be the nodes of the connected γ-clique reported by SteinerQuasi, HateSwap classifies the nodes in S (Line 3) as shown in Algorithm 7. There are five types of nodes, which are defined as follows: (1) If $u \in S$ is a query node, then u is type-1. (2) If $G[S \setminus \{u\}]$ is not connected and $H_u \cap S \neq \emptyset$, then u is type-2. (3) If $G[S \setminus \{u\}]$ is not a γ-quasi-clique and $H_u \cap S \neq \emptyset$, then u is type-3. (4) If $H_u \cap S \neq \emptyset$,

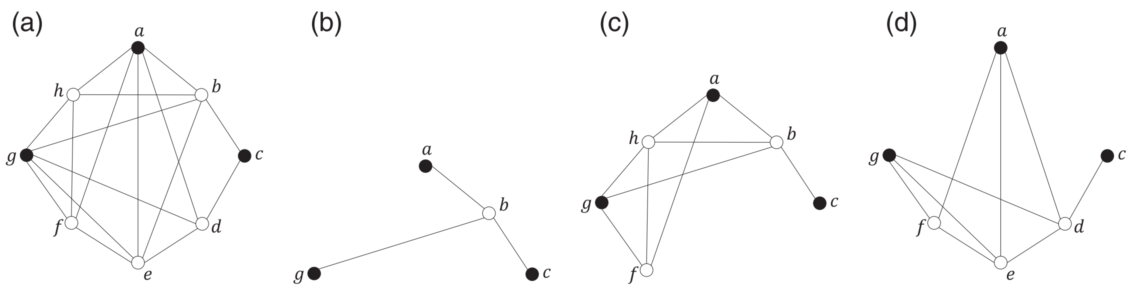

FIGURE 2. Running example of SteinerSwap.

then u is type-4. (5) Otherwise, u is type-5. The type-1 node cannot be removed since it is a query node. The type-2 and type-3 nodes can be swapped with appropriate other nodes to minimize the f value. In case of type-4 node, it can safely be removed in order to minimize the f value. The type-5 node does not need to be removed since it does not affect the f value. Type-2 and type-3 nodes are swapped with appropriate nodes in $V \setminus S$ which minimized the f values (Lines 4–7). Type-4 nodes are removed by GreedyRemove (Line 8). Finally, add the $v \in V$ to S if $H_v \cap S = \emptyset$ and $G[S \cup \{v\}]$ is a γ-clique (Lines 9 and 10). HateSwap is terminated when there does not exist any node which can be swapped or the number of iterations exceeds $I_{\max}$. HateSwap returns the set of nodes $S \subseteq V$ which is a nearly optimal solution for the FCS problem.

The time complexity of SteinerTree is $O(|Q| \times m)$. The time complexity of the connection test between query nodes in the $G[S]$ is $O(|S| + |E[S]|)$. Thus, the time complexities of SteinerQuasi and HateSwap are $O(n \times I_{\max} \times (|S| + |E[S]|))$. Therefore, the time complexity of SteinerSwap is $O(|Q| \times m + n \times I_{\max} \times (|S| + |E[S]|))$ which is significantly lower than those of Greedy, since $|S| \ll n$.

Let us explain using an example with Fig. 2. Figure 2a shows the input graph G with $Q = \{a, g, c\}$, $\gamma = 0.6$, $H_h = \{a\}$, $H_g = \{h\}$, $H_b = \{c\}$ and $\mathcal{H} = \{H_h, H_g, H_b\}$. SteinerTree outputs $T = \{a, b, c, g\}$, the set of nodes of the Steiner tree built on Q, as shown in Fig. 2b. Based on T, SteinerQuasi can find $S_0 = \{a, b, c, f, g, h\}$ such that $G[S_0]$ is a connected 0.6-clique with $f(S_0) = 3$ as shown in Fig. 2c. HateSwap classifies each node $u \in S_0$ into five types (type-1: $\{a, g, c\}$, type-2: $\{b\}$, type-3: $\{h\}$, type-5: $\{f\}$). h can be swapped with e, and b can be swapped with d. Finally, SteinerSwap reports the set $S = \{a, c, d, e, f, g\}$ with $f(S) = 0$ as the final answer.

5. EXPERIMENT

In this section, we evaluate the performance of the proposed algorithms, namely Greedy and SteinerSwap, compared with the straightforward adaptation of the existing algorithm, namely Baseline. First, we introduce our experimental setting in Section 5.1. Then, we examine the performance of algorithms by varying parameters in Sections 5.2 and 5.3.

5.1. Environment setting

To show the effectiveness of the proposed algorithms, we use three real datasets, namely Google+ (ego-Gplus), Epinions[4] (soc-Epinions) and Twitter (ego-Twitter), downloaded from SNAP.[5] Google+ and Twitter are famous social network services, and Epinions is a general consumer review site. These are well suited for our problem, since our motivating applications deal with user objects. Since bad relationships in large social networks are unknown in public, we use real datasets with synthetically generated bad relationships. We convert the directed graphs into undirected graphs by deleting the direction.

For each dataset, we generate bad relationships of each node in the real datasets under the uniform distribution. This is because information of bad relationships of a person in the real world cannot be obtained in public because of privacy issues. Moreover, the number of bad relationships of each node in the real dataset tends to be proportional to its degree, since we can consider that the number of bad relationships of a person in the real world would be proportional to the number of his/her acquaintances when there is no information. For example, consider a node u the degree of which is 100. $100 \times \alpha$ nodes, which are connected by a directed edge to u, are selected randomly. The bad relationships of u are generated based on the $100 \times \alpha$ nodes. α is the bad relationships constant. In this paper, we set α as 0.15. The details of real datasets are listed in Table 1.

For each experiment, we vary the density threshold (γ), the size of query nodes ($|Q|$) and the distance between query nodes (l). The default values of parameters are described in Table 2.

[4] http://www.epinions.com/.
[5] http://snap.stanford.edu/.

To evaluate the effectiveness of algorithms, we use $f(S)$ values, where S is the output set of nodes. Furthermore, we also use $f(S)/|S|$ values to figure out the relative amount of the nodes in bad relationships in the community. A basic intuition behind the second metric is that a large-sized community should be regarded as a more friendly community than a small-sized community when f values of their node sets are the same. In addition, we check whether our proposed algorithms can be executed in reasonable running times.

We implement all algorithms in C++. The experiments are conducted on a PC equipped with Intel Core processor i7 CPU with 3.4 GHz and 16 GB main memory.

TABLE 1. Dataset summary.

	Nodes	Edges	Diameter
ego-Gplus	107 614	13 673 453	6
soc-Epinions	75 879	508 837	14
ego-Twitter	81 306	1 768 149	7

TABLE 2. Default parameters.

Parameters	Default
γ	0.5
$\|Q\|$	3
l	2

5.2. Effectiveness of the algorithms

In this section, we show a series of experimental results by varying parameters to evaluated effectiveness of the proposed algorithms. The sets of query nodes are randomly selected. Each value in all the graphs is an average value over 50 repetitions. For all the cases, all algorithms find feasible solutions for the FCS problem.

5.2.1. Effects of the density threshold

Figure 3 shows the experimental results for varying the density threshold (γ). In all the cases, the proposed algorithms, STEINERSWAP and GREEDY, outperform BASELINE in real datasets. $f(S)$ of STEINERSWAP is on the average 16 times smaller than that of BASELINE at $\gamma = 0.6$. In terms of $f(S)/|S|$, with small $\gamma = 0.4$, $f(S)/|S|$ of STEINERSWAP is 9.5 times smaller than that of GREEDY and 18.3 times smaller than that of BASELINE. With large $\gamma = 0.6$, $f(S)/|S|$ of STEINERSWAP is 11.1 times smaller than that of GREEDY and 38.7 times smaller than that of BASELINE. Thus, the difference between $f(S)$ values of STEINERSWAP and BASELINE increases, when γ increases. The reason is that the number of nodes added to the seed set also increases as γ increases as mentioned in Section 4.2.

5.2.2. Effects of the number of query nodes

Figure 4 shows the experimental results for varying the number of query nodes ($|Q|$). Among $f(S)$ values of all algorithms, that of STEINERSWAP is the smallest. Especially,

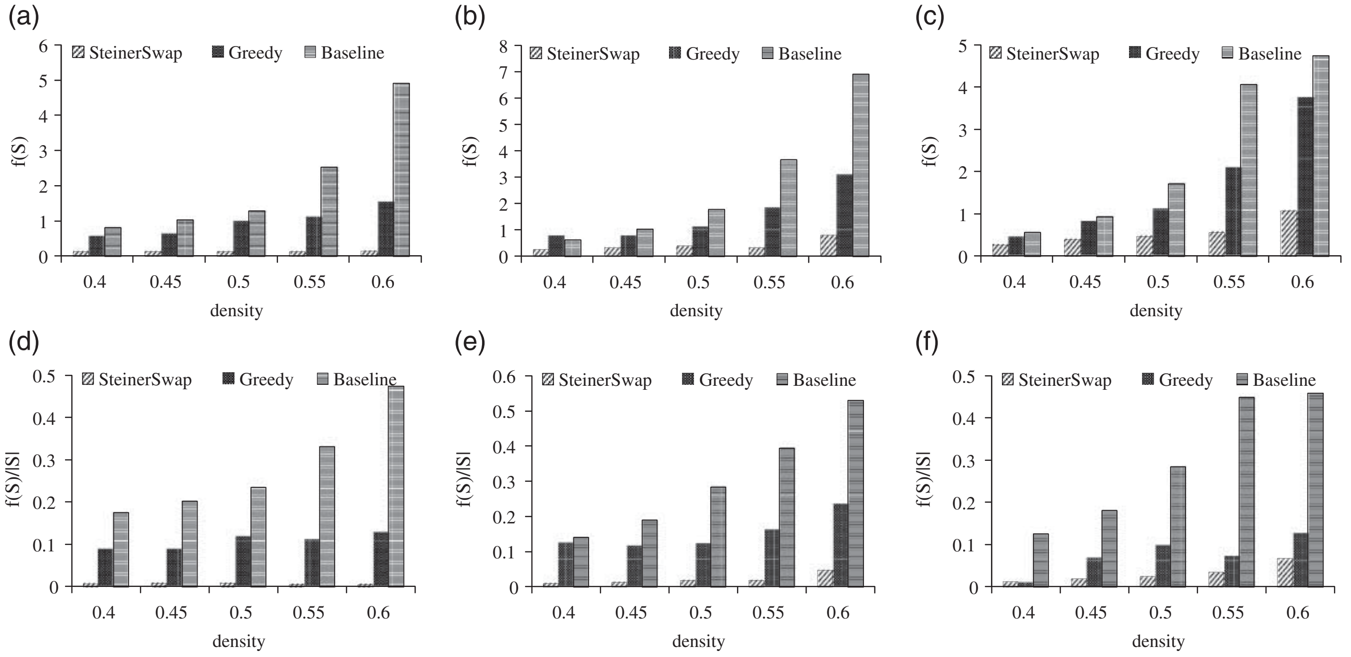

FIGURE 3. $f(S)$ and $f(S)/|S|$ values with varying γ. (**a**) ego-Gplus: $f(S)$, (**b**) soc-Epinions: $f(S)$, (**c**) ego-Twitter: $f(S)$, (**d**) ego-Gplus: $f(S)/|S|$, (**e**) soc-Epinions: $f(S)/|S|$ and (**f**) ego-Twitter: $f(S)/|S|$.

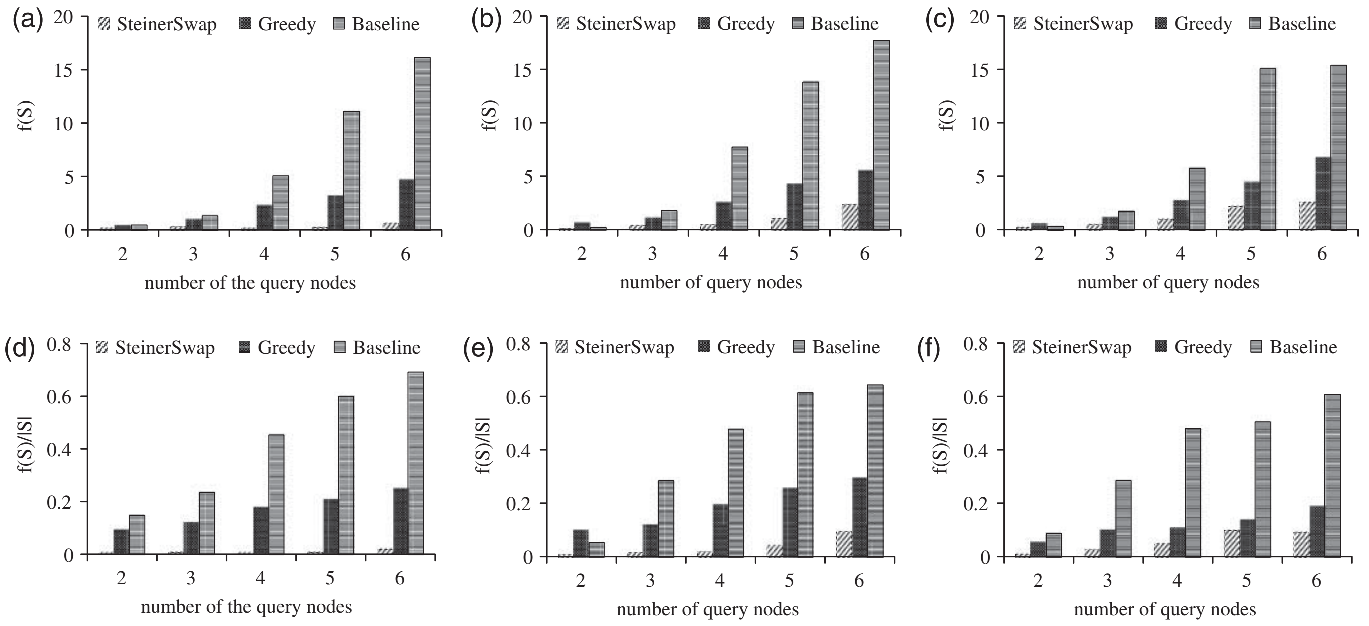

FIGURE 4. $f(S)$ and $f(S)/|S|$ values with varying $|Q|$. (**a**) ego-Gplus: $f(S)$, (**b**) soc-Epinions: $f(S)$, (**c**) ego-Twitter: $f(S)$, (**d**) ego-Gplus: $f(S)/|S|$, (**e**) soc-Epinions: $f(S)/|S|$ and (**f**) ego-Twitter: $f(S)/|S|$.

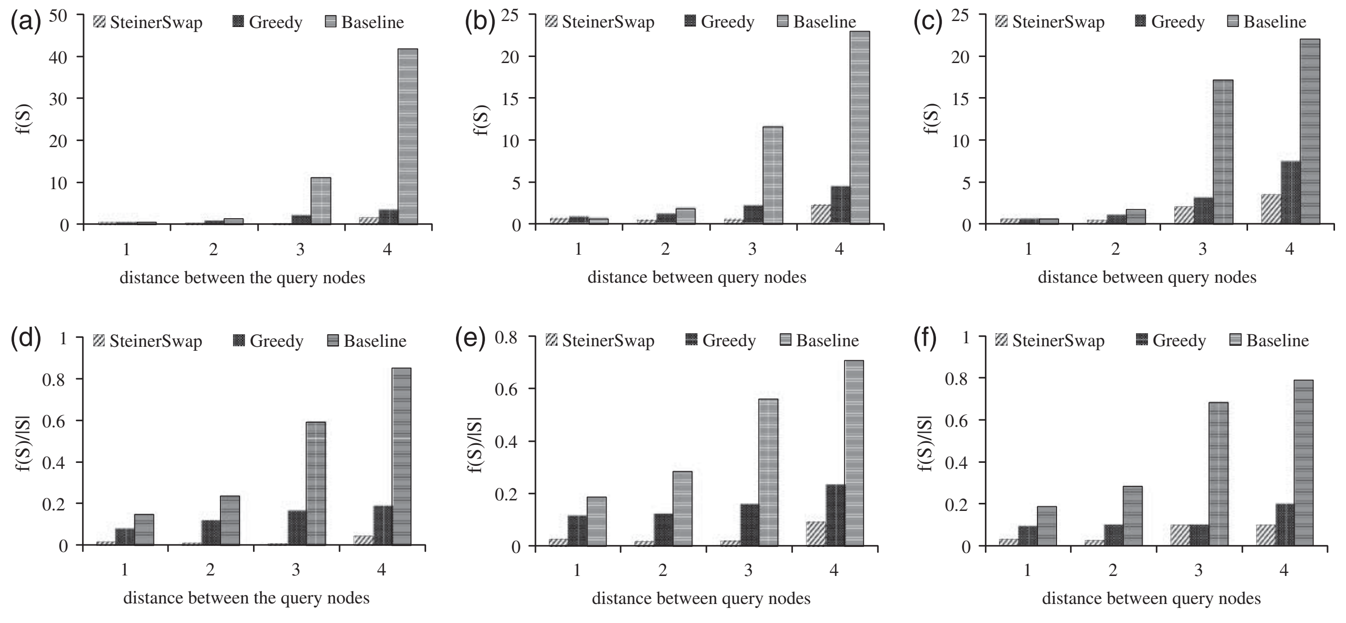

FIGURE 5. $f(S)$ and $f(S)/|S|$ values with varying l. (**a**) ego-Gplus: $f(S)$, (**b**) soc-Epinions: $f(S)$, (**c**) ego-Twitter: $f(S)$, (**d**) ego-Gplus: $f(S)/|S|$, (**e**) soc-Epinions: $f(S)/|S|$ and (**f**) ego-Twitter: $f(S)/|S|$.

$f(S)$ of SteinerSwap is up to 20.3 times smaller than that of Baseline at $|Q| = 3$. The difference between $f(S)/|S|$ values of SteinerSwap and Baseline increases, when $|Q|$ increases. With $|Q| = 3$, $f(S)/|S|$ of SteinerSwap is 14.9 times smaller than that of Greedy and 39.3 times smaller than that of Baseline. These results show that the effectiveness of Baseline gets worse when the size of the Steiner tree built on the query nodes increases as mentioned in Section 4.2. However, SteinerSwap effectively reduces the f value even though the size of the Steiner tree increases.

5.2.3. *Effects of the distance between query nodes*

Figure 5 shows the experimental results for varying the distance between query nodes (l). Similar to the above results, STEINERSWAP outperforms GREEDY and BASELINE in terms of $f(S)$ and $f(S)/|S|$. $f(S)$ of STEINERTREE is up to 30 times smaller than that of BASELINE. In addition, the difference between $f(S)/|S|$ values of STEINERSWAP and BASELINE increases, when l increases. These results show that STEINERSWAP effectively reduces the f value through swapping nodes. As the distance increases, the size of Steiner tree built on query nodes increase in order to connect query nodes. This leads to the increase of the number of nodes added as mentioned in Section 4.2.

5.3. Efficiency of the algorithms

In Section 3, we theoretically show the efficiency of the proposed algorithms in terms of the time complexity. In this section, we show the efficiency of the proposed algorithms, in terms of the running time. As mentioned earlier, each value in all the graphs is an average value over 50 repetitions. The average running times of each algorithm are summarized in Table 3.

5.3.1. *Effects of the density threshold*

Figure 6 shows the experimental results about running times for varying the density threshold (γ). It is observed that the running times of all algorithms do not depend on γ. This is because the time complexity of all algorithms do not involve the density threshold term as described in Section 4. As shown in the figure, we can see that the running time of GREEDY is the worst as we expected in Section 4.1, which is much larger than those of STEINERSWAP and BASELINE. The running time of STEINERSWAP is comparable with that of BASELINE.

5.3.2. *Effects of the number of query nodes*

Figure 7 shows the experimental results for varying the number of query nodes ($|Q|$). The running time of STEINERSWAP and BASELINE increase when $|Q|$ increases, since the time complexity of finding a Steiner tree is $O(|Q| \times n)$. However, the running time of GREEDY does not depend on the number of query nodes. This is because GREEDY removes nodes from the entire graph to find a quasi-clique. Thus, the gap between the running time of STEINERSWAP and that of GREEDY increase as $|Q|$ decreases. Nevertheless, similar to the above results, the running time of GREEDY is the worst among algorithms

TABLE 3. Running times summary.

	Running time of STEINERSWAP (s)	Running time of GREEDY (s)	Running time of BASELINE (s)
ego-Gplus	12.18	407.07	11.67
soc-Epinions	0.42	28.983	0.367
ego-Twitter	1.16	47.9	0.98

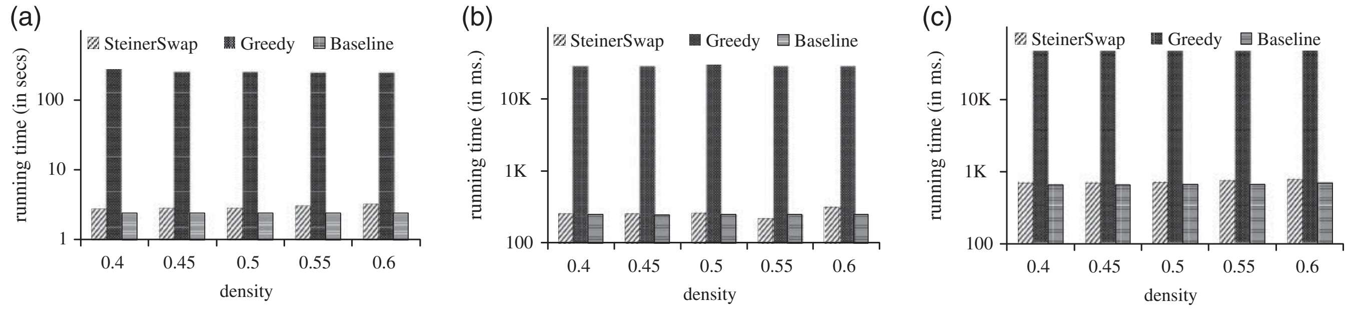

FIGURE 6. Running time with varying γ. (**a**) ego-Gplus: running time, (**b**) soc-Epinions: running time and (**c**) ego-Twitter: running time.

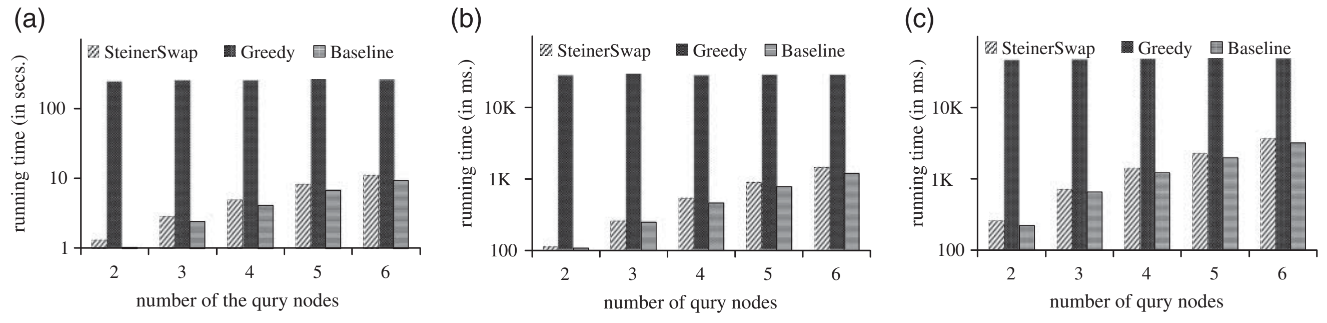

FIGURE 7. Running time with varying $|Q|$. (**a**) ego-Gplus: running time, (**b**) soc-Epinions : running time and (**c**) ego-Twitter: running time.

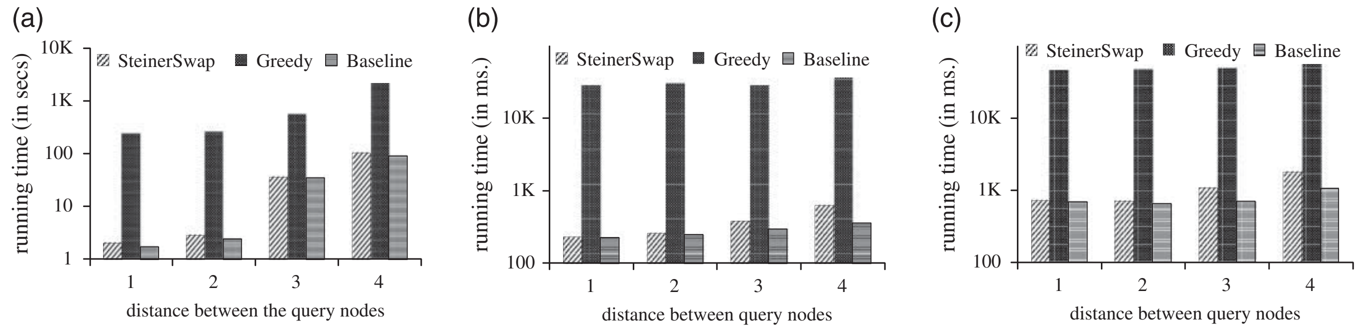

FIGURE 8. Running time with varying l. (**a**) ego-Gplus: running time, (**b**) soc-Epinions: running time and (**c**) ego-Twitter: running time.

as we expected. The running time of SteinerSwap is still comparable with that of Baseline, and much smaller than that of Greedy.

5.3.3. *Effects of the distance between query nodes*

Figure 8 shows the experimental results about running times for varying the distance between query nodes (l). As shown in the figure, the running time of SteinerSwap and that of Baseline increase when l increases. This is because the cost of finding paths between query nodes increases, when the distance between query nodes increases. However, the running time of SteinerSwap is much smaller than that of Greedy. In addition, the running time of SteinerSwap is still comparable with that of Baseline.

6. CONCLUSION

In this paper, we address the problem of finding a densely connected community containing query nodes while minimizing the number of members involved in bad relationships in the community, namely FCS. The proposed problem is useful in lots of real-world applications such as finding thematic groups and organizing social events. We prove that the proposed problem is NP-hard by using the Steiner tree problem. To solve the proposed problem, we devise two heuristic algorithms, namely Greedy and SteinerSwap. Greedy removes nodes in a given graph while SteinerSwap adds nodes to the Steiner tree built on the query nodes. We demonstrate that the proposed algorithms are effective in practice through extensive experiments on real datasets. On the average over various experimental parameters for all datasets, SteinerSwap performs 11.1 and 25.2 times better than Baseline, an adaptation of an existing algorithm, in terms of $f(S)$ and $f(S)/|S|$ values, respectively. In addition, we provide the time complexity of the SteinerSwap algorithms and that of the Greedy algorithm. We also show that the SteinerSwap algorithm is comparable with the Baseline algorithm, in terms of efficiency, through various experiments.

For the future work, we would like to extend our work to process large graphs more efficiently and to process graphs with larger density thresholds. Moreover, we plan to add a constraint on the size of an extracted community, since the size of the community is an important issue in the real world.

FUNDING

This work was supported by the National Research Foundation of Korea(NRF) Grant funded by the Korean Government(MSIP) (No. NRF-2014R1A1A2002499).

REFERENCES

[1] Girvan, M. and Newman, M.E. (2002) Community structure in social and biological networks. *Proc. Natl Acad. Sci.*, **99**, 7821–7826.

[2] Sozio, M. and Gionis, A. (2010) The Community-Search Problem and How to Plan a Successful Cocktail Party. *Proc. 16th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, Washington, DC, USA, July 25–28, pp. 939–948. ACM, USA.

[3] Sobhani, M., Fox, G.R., Kaplan, J. and Aziz-Zadeh, L. (2012) Interpersonal liking modulates motor-related neural regions. *PloS One*, **7**, e46809.

[4] Li, Y. and Shen, H. (2011) Anonymizing graphs against weight-based attacks with community preservation. *JCSE*, **5**, 197–209.

[5] Smyth, S. and White, S. (2005) A Spectral Clustering Approach to Finding Communities in Graphs. *Proc. 5th SIAM Int. Conf. Data Mining*, CA, USA, April 21–23, pp. 76–84. SIAM, USA.

[6] Newman, M.E. (2004) Fast algorithm for detecting community structure in networks. *Phys. Rev. E*, **69**, 066133.

[7] Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z. and Wagner, D. (2008) On modularity clustering. *IEEE Trans. Knowl. Data Eng.*, **20**, 172–188.

[8] Martelot, E.L. and Hankin, C. (2013) Fast multi-scale detection of relevant communities in large-scale networks. *Comput. J.*, **56**, 1136–1150.

[9] Shen, H.-W. (2013) *Community Structure of Complex Networks*. Springer, Berlin.

[10] Gómez, S., Jensen, P. and Arenas, A. (2009) Analysis of community structure in networks of correlated data. *Phys. Rev. E*, **80**, 016114.

[11] Traag, V. and Bruggeman, J. (2009) Community detection in networks with positive and negative links. *Phys. Rev. E*, **80**, 036115.

[12] Wagstaff, K. and Cardie, C. (2000) Clustering with Instance-Level Constraints. *Proc. 17th National Conf. Artificial Intelligence and 12th Conf. Innovative Applications of Artificial Intelligence*, TX, USA, July 30–August 3, vol. 1097. AAAI Press/The MIT Press, USA.

[13] Wagstaff, K., Cardie, C., Rogers, S. and Schrödl, S. (2001) Constrained k-Means Clustering with Background Knowledge. *Proc. 18th Int. Conf. Machine Learning*, MA, USA, June 28–July 1, pp. 577–584. Morgan Kaufmann, USA.

[14] Basu, S., Banerjee, A. and Mooney, R.J. (2004) Active Semi-Supervision for Pairwise Constrained Clustering. *Proc. 4th SIAM Int. Conf. Data Mining*, USA, April 22–24, pp. 333–344. SIAM, USA.

[15] Bilenko, M., Basu, S. and Mooney, R.J. (2004) Integrating Constraints and Metric Learning in Semi-Supervised Clustering. *Machine Learning, Proc. 21st Int. Conf.*, Alberta, Canada, July 4–8, pp. 595–602. ACM, USA.

[16] Lappas, T., Liu, K. and Terzi, E. (2009) Finding a Team of Experts in Social Networks. *Proc. 15th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, Paris, France, June 28–July 1, pp. 467–476. ACM, USA.

[17] Anagnostopoulos, A., Becchetti, L., Castillo, C., Gionis, A. and Leonardi, S. (2012) Online Team Formation in Social Networks. *Proc. 21st World Wide Web Conf. 2012*, Lyon, France, April 16–20, pp. 839–848. ACM, USA.

[18] Majumder, A., Datta, S. and Naidu, K.V.M. (2012) Capacitated Team Formation Problem on Social Networks. *The 18th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, Beijing, China, August 12–16, pp. 1005–1013. ACM, USA.

[19] Kargar, M. and An, A. (2011) Discovering Top-k Teams of Experts with/without a Leader in Social Networks. *Proc. 20th ACM Conf. Information and Knowledge Management*, Glasgow, UK, October 24–28, pp. 985–994. ACM, USA.

[20] Li, C.-T. and Shan, M.-K. (2012) Composing Activity Groups in Social Networks. *21st ACM Int. Conf. Information and Knowledge Management*, Maui, HI, USA, October 29–November 2, pp. 2375–2378. ACM, USA.

[21] Gallo, G., Grigoriadis, M.D. and Tarjan, R.E. (1989) A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.*, **18**, 30–55.

[22] Charikar, M. (2000) Greedy Approximation Algorithms for Finding Dense Components in a Graph. *Approximation Algorithms for Combinatorial Optimization, 3rd Int. Workshop*, Saarbrücken, Germany, September 5–8, pp. 84–95. Springer, USA.

[23] Uno, T. (2010) An efficient algorithm for solving pseudo clique enumeration problem. *Algorithmica*, **56**, 3–16.

[24] Tsourakakis, C.E., Bonchi, F., Gionis, A., Gullo, F. and Tsiarli, M.A. (2013) Denser than the Densest Subgraph: Extracting Optimal Quasi-Cliques with Quality Guarantees. *Proc. 19th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, Chicago, IL, USA, August 11–14, pp. 104–112. ACM, USA.

[25] Karp, R.M. (1972) Reducibility Among Combinatorial Problems. *Proc. Symp. Complexity of Computer Computations*, New York, USA, March 20–22, pp. 85–103. Plenum Press, USA.

[26] Pardalos, P.M., Pardalos, P.M. and Pardalos, P.M. (1993) *Complexity in Numerical Optimization*. World Scientific, Singapore.

[27] Kou, L.T., Markowsky, G. and Berman, L. (1981) A fast algorithm for Steiner trees. *Acta Inf.*, **15**, 141–145.

APPENDIX. PROOF OF THEOREM 4.1

Proof. Let S be a subset of nodes. Edge surplus h_γ is defined as $h_\gamma(S) = |E[S]| - \gamma\binom{|S|}{2}$. Let $d_S(u)$ denote the degree of the node $u \in S$ in $G[S]$. Note that $h_\gamma(S^*) \geq 0$ and $S^* \subseteq S_I$.

$$\begin{aligned}
d_{S_I}(u) &\geq d_{S^*}(u) \geq 2, \forall u \in (S^* \setminus Q)\\
h_\gamma(S_I) &= |E[S_I]| - \gamma\binom{|S_I|}{2}\\
&= \frac{1}{2}\sum_{u\in S_I} d_{S_I}(u) - \gamma\binom{|S_I|}{2}\\
&= \frac{1}{2}\left(\sum_{u\in S^*} d_{S^*}(u) + \sum_{u\in S^*}(d_{S_I}(u) - d_{S^*}(u))\right.\\
&\quad \left. + \sum_{u\in S_I\setminus S^*} d_{S_I}(u)\right) - \gamma\binom{|S_I|}{2}\\
&\geq \frac{1}{2}\left(\sum_{u\in S^*} d_{S^*}(u) + \sum_{u\in S_I\setminus S^*} d_{S_I}(u)\right) - \gamma\binom{|S_I|}{2}\\
&= |E[S^*]| + \frac{1}{2}\sum_{u\in S_I\setminus S^*} d_{S_I}(u) - \gamma\binom{|S_I|}{2}\\
&\geq |E[S^*]| + (|S_I| - |S^*|) - \gamma\binom{|S_I|}{2}\\
&= |E[S^*]| - \gamma\binom{|S^*|}{2} + \gamma\binom{|S^*|}{2}\\
&\quad + (|S_I| - |S^*|) - \gamma\binom{|S_I|}{2}\\
&\geq \gamma\binom{|S^*|}{2} + (|S_I| - |S^*|) - \gamma\binom{|S_I|}{2}\\
&\Leftrightarrow h_\gamma(S_I) \geq \gamma\binom{|S^*|}{2} + (|S_I| - |S^*|) - \gamma\binom{|S_I|}{2}\\
&\Leftrightarrow |E[S_I]| \geq \gamma\binom{|S^*|}{2} + (|S_I| - |S^*|)\\
&\Leftrightarrow \frac{|E[S_I]| - (|S_I| - |S^*|)}{\binom{|S^*|}{2}} \geq \gamma.
\end{aligned}$$

□