Knowl Inf Syst (2015) 45:617–643
DOI 10.1007/s10115-014-0800-9

REGULAR PAPER

Triangle minimization in large networks

Rong-Hua Li · Jeffrey Xu Yu

Received: 6 November 2013 / Revised: 14 July 2014 / Accepted: 29 October 2014 /
Published online: 4 December 2014

Abstract The number of triangles is a fundamental metric for analyzing the structure and function of a network. In this paper, for the first time, we investigate the triangle minimization problem in a network under edge (node) attack, where the attacker aims to minimize the number of triangles in the network by removing k edges (nodes). We show that the triangle minimization problem under edge (node) attack is a submodular function maximization problem, which can be solved efficiently. Specifically, we propose a degree-based edge (node) removal algorithm and a near-optimal greedy edge (node) removal algorithm for approximately solving the triangle minimization problem under edge (node) attack. In addition, we introduce two pruning strategies and an approximate marginal gain evaluation technique to further speed up the greedy edge (node) removal algorithm. We conduct extensive experiments over 12 real-world datasets to evaluate the proposed algorithms, and the results demonstrate the effectiveness, efficiency and scalability of our algorithms.

Keywords Triangle minimization · Large networks · FM sketch · Submodular function

1 Introduction

A network comprises a set of nodes and a set of edges representing the connections between the nodes. In the past decade, network science has attracted growing attention in both industry and research communities due to a large number of applications. Many real-world problems can be modeled as a network. For example, in online social networks, a user can be modeled by a node and a social tie between two users can be represented by an edge. In computer networks, every computer can be modeled by a node and each link between two computers

R.-H. Li (✉)
Guangdong Province Key Laboratory of Popular High Performance Computers,
Shenzhen University, Shenzhen, China
e-mail: lironghuascut@gmail.com

J. X. Yu
The Chinese University of Hong Kong, Shatin, Hong Kong

can be represented by an edge. Another example is the Internet, where each web page can be represented by a node and each hyperlink signifies an edge. As observed by Watts and Strogatz [50], many real-world networks exhibit small world phenomenon that the diameter of the network is extremely small regarding to the network size. Subsequently, Barabási and Albert [5] reported that most real-world networks also exhibit power-law degree distribution, where the degree of the most nodes in a network is relatively small and only a few nodes have large degree. These two crucial discoveries inspire comprehensive research on network science. The studies on network science mainly address the structure and function of a network.

One important topic of network science is to study the error and attack tolerance of a network [1,11,32,33,42]. As discussed in [9,11], many real-world networks are vulnerable to targeted attacks. In general, there are two types of attacks in a network: edge attack and node attack. For edge attack, the attacker aims to disconnect certain edges of the network, while for node attack, the goal of the attacker is to remove some nodes from the network. For example, a recent news in CNET[1] reports an edge attack event. The fast-food company Burger King created a Facebook application, namely Whopper Sacrifice, which will give a Facebook user a free coupon for a free hamburger if he/she deletes ten people from his/her friends list. By statistics, 82,771 Facebook users participated in this campaign, and 233,906 Facebook friendships were deleted. After then, Facebook stops this application due to a large number of link deletions. The node attacks frequently happen to computer networks. For instance, a hacker attacks a server in the network and makes it break down. Clearly, this attack is equivalent to removing a node from the network, thereby it is a node attack.

In practice, for edge (node) attack, the attacker, due to resource limitation, has a small budget of k edges (nodes) to attack. We assume that if an edge (a node) is attacked by the attacker, then the edge (node) will be removed from the network. Generally, the attacker wants to maximize some utility functions by attacking k edges (nodes). Previous studies have been addressed to such edge and node attacks. For example, in Albert et al. [1], study the diameter of the resulting network after k edges removal. In [1], the utility function of the attacker corresponds to the diameter of the network, and the attacker aims to maximize the diameter of the network, which will make the network as incohesive as possible. In Schneider et al. [42], study the minimal size of the maximal connected component (MCC) of the remaining network after k nodes deletions. Based on this, they define a metric for measuring the robustness of a network. In [42], the utility function of the attacker corresponds to the inverse of the size of the maximal connected component of the network, and the attacker aims to minimize the size of the MCC of the network, which will reduce the robustness of the network as much as possible. More recently, Tong et al. [45] investigate a problem of deleting k edges to minimize the leading eigenvalue of the adjacency matrix of the graph. Clearly, in their work, the utility function of the attacker is the inverse of the leading eigenvalue.

In contrast to the diameter, the size of the maximal connected component and the leading eigenvalue of a network, in this paper, we consider another important metric, i.e., the number of triangles of the network. It is well known that triangles play an crucial role in social network analysis [18]. The theories of homophily [36] and transitivity in social network analysis are based on the triangles of the social network. In particular, by the homophily theory, users in a social network tend to select friends who are similar to themselves [36]. In terms of the transitivity theory, users who have common friends tend to be friends themselves [18]. Due to the significance of the triangles in social network analysis, a lot of metrics, such as the number of triangles, clustering coefficient [50] and transitivity ratio [18], are proposed for triangle analysis in social networks [18]. Among them, the number of triangles is a very basic

[1] http://news.cnet.com/delete-10-facebook-friends-get-a-free-whopper/.

metric, and it is closely related to the other metrics. Specifically, the number of triangles is proportional to the clustering coefficient and transitivity ratio of a network. In general, the tighter, the community, the more likely it has a large number of triangles. A community with a large number of triangles typically exhibits many important functions (e.g., homophily phenomenon, transitivity and the formation of positive social norms [12]). However, in many real-world applications, the network could be attacked by malicious individuals, which results in edge or node removal. It is therefore crucial to study how the number of triangles of a network changes after removing a certain number of edges or nodes.

To address this problem, in this paper, we study two triangle minimization problems in a network subject to two different attack models, which are edge attack and node attack, respectively. To the best of our knowledge, we are the first group to study the triangle minimization problems in networks. Specifically, in our problems, the attacker aims to minimize the number of triangles of the resulting network by attacking k edges (edge attack model) or k nodes (node attack model). Unlike the diameter, the size of the MCC and the leading eigenvalue of a network, we show that the triangle minimization problems can be approximately solved by an efficient greedy algorithm via exploiting the submodularity property. Additionally, we emphasize that the solutions of triangle minimization problems can be used in two different scenarios. On the one hand, from the attacker's perspective, the attackers can use the proposed method to attack a network so as to achieve maximal utilities. On the other hand, from the network administrator's perspective, the network administrators can protect some "important" edges (or nodes) identified by our methods so as to defend attacks.

The main contributions of this paper are summarized as follows. First, we introduce the triangle minimization problem in a network under both edge and node attacks. For edge (node) attack, we formulate the problem as a submodular function maximization problem and show that the problem is NP-hard. Second, to approximately solve the problem, we propose a degree-based edge (node) removal algorithm and an efficient greedy edge (node) removal algorithm with near optimal performance guarantee ($\approx$.0.63 approximation factor). For both greedy edge removal algorithm and greedy node removal algorithm, we propose two pruning strategies to accelerate the algorithms. The time complexity of the degree-based edge (node) removal algorithm is linear with respect to the size of the network, and the time complexity of the greedy edge (node) removal algorithm is $O(km^{1.5})$, where m is the number of edges of the network and $O(m^{1.5})$ is the time complexity for triangle counting. To further reduce the time complexity of the greedy algorithms, we propose a near-optimal linear time greedy algorithm based on a well-known probabilistic counting structure Flajolet–Martin (FM) sketch [16] where the FM sketch is utilized to approximately estimate the marginal gain. We show that the space complexity of all the proposed algorithms is linear with respect to the network size. Moreover, all of our algorithms are easy to implement. Finally, we conducted extensive experimental studies over 12 real-world networks. The results demonstrate the effectiveness, efficiency and scalability of the proposed algorithms. Furthermore, we show that the proposed greedy algorithms are also very effective to decrease the clustering coefficient of a network, suggesting that our greedy algorithms are also good heuristic algorithms to minimize the clustering coefficient of a network under edge (node) attack.

The rest of this paper is organized as follows. We give the problem statement in Sect. 2. We propose the degree-based edge removal algorithm and the greedy edge removal algorithm, and the degree-based node removal algorithm and the greedy node removal algorithm in Sects. 3 and 4, respectively. Extensive experiments are reported in Sect. 5, and the related work is discussed in Sect. 6. We conclude this work in Sect. 7.

2 Problem statement

We model a network by an undirected and unweighted graph $G = (V, E)$, where V signifies a set of nodes and E denotes a set of undirected edges. Let $n = |V|$ and $m = |E|$ be the number of nodes and the number of edges in G, respectively. Given a graph G, the problem that we address in this paper is to minimize the number of triangles in G given a budget of k edges/nodes removal. As mentioned in the introduction, our problem is motivated by the following attack model under network setting.

We assume that there is an attacker who has a budget of k edges (nodes) to attack. If an edge (a node) is attacked by the attacker, then the edge (node) will be deleted from the network. The goal of the attacker is to minimize the number of triangles in the network after removing k edges/nodes. For convenience, we refer to the attack model based on k edges and k nodes removal as the edge attack model and the node attack model, respectively. Below, we define our problems based on these two attack models.

First, we consider the triangle minimization problem based on edge attack model. Let $A \in \mathbb{R}^{n\times n}$ be the adjacency matrix of the graph G, $A(i, j)$ be the element in the ith row and jth column of A, and $tr(A)$ be the trace of the matrix A. Clearly, for each edge $e = (u, v) \in E$, we have $A(u, v) = 1$. Further, we let S be a subset of edges, i.e., $S \subseteq E$, and $|S|$ be the cardinality of the set S. And let $B_S \in \mathbb{R}^{n\times n}$ be a matrix such that $B_S(u, v) = 1$ if $(u, v) \in S$ and $B_S(u, v) = 0$ otherwise. If S only contains one edge e, we simply denote the matrix B_S as B_e.

Based on these notations, we first describe a well-known lemma in algebraic graph theory [17], and then define our problem.

Lemma 2.1 *Given a undirected and unweighted graph G and its adjacency matrix A. The number of triangles in G is equal to the trace of A^3 divided by six, i.e., $tr(A^3)/6$.*

Based on Lemma 2.1, our problem can be formulated as a discrete optimization problem as follows.

Problem 1 Given a graph G, its adjacency matrix A, and a budget size k. Then, the triangle minimization problem under the edge attack model can be formulated as

$$\begin{aligned} &\min \quad tr((A - B_S)^3) \\ &\text{s.t.} \quad |S| \leq k, \end{aligned} \tag{1}$$

where S is a subset of edges, i.e., $S \subseteq E$. For any subset S, we define a function $F(S)$ as $F(S) = tr(A^3) - tr((A - B_S)^3)$. It is very easy to verify that the problem defined in Eq. (1) is equivalent to

$$\begin{aligned} &\max \quad F(S) \\ &\text{s.t.} \quad |S| \leq k. \end{aligned} \tag{2}$$

The following theorem shows that the problem defined in Eq. (2) is NP-hard.

Theorem 2.1 *For a general undirected graph G, the optimization problem defined in Eq. (2) is NP-hard.*

Proof See "Appendix". □

Second, in a similar manner, we can formulate the triangle minimization problem based on the node attack model. Let X be a subset of nodes, i.e., $X \subseteq V$, and $C_X \in \mathbb{R}^{n\times n}$ be a

matrix such that $C_X(u, v) = 1$ if $u \in X$ and $(u, v) \in E$ or $v \in X$ and $(u, v) \in E$, and $C_X(u, v) = 0$ otherwise. If X only consists of one node v, then we denote C_X as C_v. Based on these notations, we describe our problem in Problem 2.

Problem 2 Given a graph G, its adjacency matrix A, and a budget size k. The triangle minimization problem based on the node attack model can be formulated as

$$\begin{aligned} \max \quad & J(X) \\ \text{s.t.} \quad & |X| \leq k, \end{aligned} \tag{3}$$

where $J(X) = tr(A^3) - tr((A - C_X)^3)$. Note that $tr((A - C_X)^3)$ denotes the number of triangles in the residual graph after removing the nodes in X. Likewise, Problem 2 is also NP-hard.

Theorem 2.2 *For a general undirected graph G, the optimization problem defined in Eq.* (3) *is NP-hard.*

Proof See "Appendix". □

Given the hardness of our two problems, we resort to design heuristic algorithms for approximately solving them. In the next two sections, we will devise several heuristic algorithms for Problems 1 and 2, respectively.

3 Algorithms for edge removals

As discussed in the previous section, both of Problems 1 and 2 are NP-hard, thus we cannot develop an algorithm to solve them in polynomial time unless $P = \text{NP}$. In this section, we concentrate on devising practical algorithms for Problem 1. In particular, we first present a degree-based edge removal algorithm, which is very efficient. But unfortunately, the degree-based algorithm is without performance guarantee. To tackle this issue, we propose a greedy edge removal algorithm, which is shown to be near-optimal ($1 - 1/e$ approximation). Moreover, we also present two pruning techniques to further accelerate the greedy edge removal algorithm. The detailed descriptions of these algorithms are given in the following subsections.

3.1 Degree-based edge removal algorithm

First, we define a degree function for each edge $e = (u, v)$ in E as $d(e) = d(u, v) = \min\{d_u, d_v\}$, where d_u and d_v denote the degree of node u and v, respectively. Then, our algorithm iteratively removes k edges with maximal $d(e)$. We describe our algorithm in Algorithm 1. First, Algorithm 1 computes $d(e)$ for all the edges in G (line 1). Second, the algorithm works in k rounds, and at each round, the algorithm finds an edge e with maximal $d(e)$ denoted as $e_{\max}$ (line 4). Then, the algorithm adds the edge $e_{\max}$ into the answer set S and deletes it from E (line 5–6). Let u and v be the two end nodes of $e_{\max}$. Then, the algorithm decreases the degree of u and v by 1 (line 7–8) and recomputes $d(e)$ for all the incident edges of u and v, because only the degree function $d(e)$ of those edges may need to be updated.

The time complexity of Algorithm 1 is $O(km)$. First, computing $d(e)$ for all the edges takes $O(m)$ time. Second, at each round, finding the edge $e_{\max}$ consumes $O(m)$ time and the time complexity for updating $d(e)$ for all the incident edges of the end nodes is dominated by

Algorithm 1 Degree-based edge removal algorithm

Input: Graph $G = (V, E)$ and k
Output: A set S with k edges

1: Compute $d(e)$ for each edge e
2: $S \leftarrow \emptyset$
3: **for** $i = 1$ to k **do**
4: Find $e_{\max} = (u, v) = \arg\max_{e \in (E \setminus S)} d(e)$
5: $S \leftarrow S \bigcup \{e_{\max}\}$
6: Delete $e_{\max}$ from E
7: $d_u \leftarrow d_u - 1$
8: $d_v \leftarrow d_v - 1$
9: Update $d(e)$ for all the incident edges of node u and v
10: **return** S

Algorithm 2 Greedy edge removal algorithm

Input: Graph $G = (V, E)$ and k
Output: A set S with k edges

1: $S \leftarrow \emptyset$
2: **for** $i = 1$ to k **do**
3: $e_{\max} \leftarrow NULL$
4: $\Delta_{\max} \leftarrow -\infty$
5: **for** each edge e in $G = (V, E \setminus S)$ **do**
6: Compute Δ_e in $G = (V, E \setminus S)$
7: **if** $\Delta_e > \Delta_{\max}$ **then**
8: $\Delta_{\max} \leftarrow \Delta_e$
9: $e_{\max} \leftarrow e$
10: $S \leftarrow S \bigcup \{e_{\max}\}$
11: Delete $e_{\max}$ from E
12: **return** S

$O(m)$. Put it all together, the time complexity of Algorithm 1 is $O(km)$. The space complexity of Algorithm 1 is $O(m + n)$. The reason is because we only need to store the graph as well as two additional arrays $d(e)$ and d_u, which are dominated by $O(m + n)$.

3.2 Greedy edge removal algorithm

Recall that our intention is to minimize the number of triangles after deleting k edges. Intuitively, a better way to achieve this goal is to delete k edges that are contained in as many triangles as possible. Based on this intuition, here we propose a greedy edge removal algorithm. Let $\Delta_e = \Delta(u, v)$ be the number of triangles that contain edge $e = (u, v)$. Unlike the degree-based edge removal algorithm, the greedy edge removal algorithm iteratively deletes k edges with maximal Δ_e. We outline our algorithm in Algorithm 2. Specifically, Algorithm 2 first initializes the answer set S to be an empty set. Second, Algorithm 2 removes k edges in k rounds. At each round, the algorithm iteratively finds the edge $e_{\max}$ with maximal Δ_e in the graph $G = (V, E \setminus S)$ (line 5–9). Then, the algorithm adds the edge $e_{\max}$ into the answer set S and then deletes it from E (line 10–11).

We analyze the time complexity of Algorithm 2 as follows. First, at each round, the time complexity for computing Δ_e for every edge in $G = (V, E \setminus S)$ is $O(m^{1.5})$, which can be achieved by the fast main-memory triangle counting algorithm [26]. Second, after computing Δ_e for all the edges in $G = (V, E \setminus S)$, the time complexity for finding the edge $e_{\max}$ is $O(m - |S|)$. Therefore, the total time complexity of Algorithm 2 is $O(km^{1.5})$. The

space complexity of Algorithm 2 is $O(m+n)$. The rationale is that we only need to store the graph and maintain Δ_e for every edge $e \in E$, which are dominated by $O(m+n)$.

3.3 Justification

In this subsection, we present theoretical justifications for Algorithms 1 and 2, respectively. First, we give a simple interpretation for Algorithm 1 based on the following theorem.

Theorem 3.1 $\Delta_e \leq d(e)$.

Proof Let $e = (u, v)$ be an edge in G. Since $d(e) = \min\{d_u, d_v\}$, the number of common neighbors of node u and v is bounded by $d(e)$. Note that the number of common neighbors of node u and v is equal to the number of triangles that contain e. Therefore, we have $\Delta_e \leq d(e)$. □

Based on Theorem 3.1, the degree-based edge removal algorithm (Algorithm 1) can be interpreted as deleting the edge to the current graph that minimizes the upper bound on the number of triangles of the resulting graph.

Second, we present a theoretical justification for Algorithm 2 by exploiting the submodularity of the function $F(S)$. Below, we first introduce the definition of nondecreasing submodular set function. Let U be a finite set. A set function f defined on the subsets of U is a nondecreasing submodular function if the following condition holds. For any two subsets Y and Z such that $Y \subseteq Z \subseteq U$, and for any element $j \notin Z$, we have $\rho_j(Y) \geq \rho_j(Z) \geq 0$, where $\rho_j(Y)$ signifies the marginal gain defined as $\rho_j(Y) = f(Y \cup \{j\}) - f(Y)$. Then, the following theorem shows that $F(S)$ defined in Sect. 2 is a nondecreasing submodular function.

Theorem 3.2 *$F(S) = tr(A^3) - tr((A - B_S)^3)$ is a nondecreasing submodular function with $F(\emptyset) = 0$.*

Proof First, $F(\emptyset) = 0$ obviously holds, because $B_\emptyset$ is a $n \times n$ zero matrix. Second, we prove that $F(S)$ is nondecreasing. Let $S \subseteq T \subseteq E$ be two subsets of E, and $e \notin T$ is an edge in E. Further, let $\rho_e(S) = F(S \bigcup \{e\}) - F(S)$ be the marginal gain. Note that the nondecreasing property of function $F(S)$ can be guaranteed by $\rho_e(T) \geq 0$. Below, we show that $\rho_e(T) \geq 0$. By definition, we have

$$\begin{aligned}
\rho_e(T) &= tr((A - B_T)^3) - tr((A - B_{T\cup\{e\}})^3) \\
&= tr((A - B_T)^3 - (A - B_T - B_e)^3) \\
&= 3tr((A - B_T)^2 B_e) - 3tr(B_e^2(A - B_T)) + tr(B_e^3) \\
&= 3tr((A - B_T - B_e)B_e(A - B_T)) + tr(B_e^3) \\
&\geq 0. \qquad (4)
\end{aligned}$$

The last two equalities hold due to an important property of trace, which is invariant under cyclic permutations, i.e., $tr(ABC) = tr(BCA) = tr(CAB)$ for any $n \times n$ matrices A, B, and C. Since $A - B_T$, B_e, and $A - B_T - B_e$ are nonnegative matrices, the trace of their product is also nonnegative. Therefore, the last inequality holds. Similarly, we have $\rho_e(S) = 3tr((A - B_S - B_e)B_e(A - B_S)) + tr(B_e^3) \geq 0$. Finally, we prove the submodularity of $F(S)$, i.e., $\rho_e(S) \geq \rho_e(T)$. Using the result in Eq. (4), we have

$$
\begin{aligned}
&\rho_e(S) - \rho_e(T) \\
&= 3tr((A - B_S - B_e)B_e(A - B_S)) \\
&\quad -3tr((A - B_T - B_e)B_e(A - B_T)) \\
&= 3tr((A - B_T - B_e + B_T - B_S)B_e(A - B_T + B_T - B_S)) \\
&\quad -3tr((A - B_T - B_e)B_e(A - B_T)) \\
&= 3tr(((A - B_T - B_e)B_e + (B_T - B_S)B_e)(A - B_T \\
&\quad +B_T - B_S)) - 3tr((A - B_T - B_e)B_e(A - B_T)) \\
&= 3tr((A - B_T - B_e)B_e(B_T - B_S)) \\
&\quad +3tr((B_T - B_S)B_e(A - B_S)) \\
&\geq 0,
\end{aligned}
$$

where the last inequality holds as the matrices $A - B_T - B_e$, B_e, $B_T - B_S$, and $A - B_S$ are nonnegative. This completes the proof. □

Based on Theorem 3.2, Problem 1 is a nondecreasing submodular function maximization subject to cardinality constraint problem. By a celebrated result in [38], there is a near-optimal greedy algorithm for solving such problem. The greedy algorithm iteratively selects an element in the ground set with maximal marginal gain. In the following, we show that Algorithm 2 is an instance of such greedy algorithm. Formally, we let $\Delta_e(S)$ be the number of triangles in the graph $G = (V, E\backslash S)$. The following theorem shows that $\Delta_e(S) = \rho_e(S)/6$.

Theorem 3.3 $\Delta_e(S) = \rho_e(S)/6$.

To prove Theorem 3.3, we first give some useful lemmas as follows. The proofs can be easily obtained, thus we omit for brevity.

Lemma 3.1 $tr(B_e) = 0$, *and* $B_e^3 = B_e$.

Lemma 3.2 *For any* $n \times n$ *matrix* H, *we have* $tr(B_e^2 H) = H(u, u) + H(v, v)$, *where* $e = (u, v)$.

Lemma 3.3 *Let* $e = (u, v)$. *Then, for any* $n \times n$ *symmetric matrix* H, *we have* $tr(B_e H) = 2H(u, v)$.

Armed with the above lemmas, we prove the Theorem 3.3 as follows.

Proof of Theorem 3.3 According to Eq. (4), we have

$$
\begin{aligned}
\rho_e(S) &= 3tr((A - B_S - B_e)B_e(A - B_S)) + tr(B_e^3) \\
&= 3tr(B_e^2(A - B_S)) + 3tr(B_e(A - B_S)^2) \\
&= 3tr(B_e(A - B_S)^2). \qquad (5)
\end{aligned}
$$

In the above equation, the second equality holds due to Lemma 3.1, and the last equality holds due to Lemma 3.2 and the element of the uth row and uth column as well as the element of the vth row and vth column of matrix $(A - B_S)$ equal to 0. Let $H = (A - B_S)$. Clearly, both H and H^2 are symmetric. Then, by Lemma 3.3, we have $\rho_e(S) = 6H^2(u, v)$. By definition, the matrix $H = (A - B_S)$ is an adjacency matrix of the graph $G = (V, E\backslash S)$. Then, $H^2(u, v) = \sum_{w=1}^{n} H(u, w)H(w, v)$. Note that $\sum_{w=1}^{n} H(u, w)H(w, v)$ denotes the number of triangles in graph $G = (V, E\backslash S)$ that contain edge (u, v), which is equal to $\Delta_e(S)$. Hence, we have $\Delta_e(S) = \rho_e(S)/6$. This completes the proof. □

Based on Theorem 3.3, we can conclude that Algorithm 2 iteratively finds the edge with maximal marginal gain. By a well-known result in [38], Algorithm 2 achieves a $1 - 1/\mathbf{e}$ approximate factor as stated in the following theorem, where $\mathbf{e} = 2.718$ denotes the Euler's number.

Theorem 3.4 *Algorithm 2 is a* $1 - 1/\boldsymbol{e}$ *approximate algorithm for Problem 1.*

Proof We can prove it using a similar argument presented in [38], thus we omit for brevity. □

We remark that the submodular function maximization with cardinality constraint problem is known to be non-approximate in $1 - 1/\mathbf{e} + \epsilon$ where ϵ is a very small constant [14,49]. Since our problem is an instance of submodular function maximization subject to cardinality constraint problem, thus there is no algorithm that can beat the proposed one.

3.4 Pruning techniques

Recall that Algorithm 2 needs to evaluate Δ_e for each edge e in graph $G = (V, E \backslash S)$ at each round, which is expensive. To further speed up Algorithm 2, here we introduce two pruning techniques without loss of approximation factor of the algorithm. The first pruning technique is based on the so called *lazy evaluation* strategy [28,37], and the second pruning technique is based on the upper bound developed in Theorem 3.1. We refer to the greedy algorithm with these two pruning strategies as the accelerated greedy edge removal algorithm.

Lazy evaluation (CELF optimization, [28,37]*)* The lazy evaluation is based on the submodularity property of the objective function $F(S)$, which is also called CELF optimization in [28]. Specifically, let $\emptyset \subset S_1 \subset, \cdots, \subset S_k$ be the optimal edge set found by Algorithm 2 at iteration $i = 0, 1, \ldots, k$, respectively. The key idea of CELF optimization is that at round i, the algorithm does not need to evaluate $\Delta_e(S_i)$ for those edges that their marginal gain at round $i - 1$ ($\Delta_e(S_{i-1})$) are smaller than the current maximum marginal gain ($\Delta_{e_{\max}}(S_i)$). This idea can be implemented by a priority queue. In particular, we set the priority of each edge by its marginal gain. At each round, we get the top element from the priority queue and recompute its marginal gain. Then, we update the priority of the top element by the newly computed marginal gain and then record the current maximal marginal gain as well as the corresponding edge ($e_{\max}$). If the current maximal marginal gain is greater than or equal to the current maximal priority of the queue, then we are done. Because we do not need to update the marginal gain of the remaining edges in the queue based on the property $\Delta_e(S_i) \leq \Delta_e(S_{i-1}) \leq \Delta_{e_{\max}}(S_i)$.

Further lazy evaluation (degree-based pruning) According to Theorem 3.1, for each edge e, we have $d(e) \geq \Delta_e$. At each round, we first compute $d(e)$ for each edge $e \in E \backslash S$. If $d(e)$ is smaller than the current maximal marginal gain, we do not need to recompute Δ_e because $\Delta_e \leq d(e) \leq \Delta_{\max}$. This idea can be combined with CELF optimization described above. We present our algorithm that combines these two pruning strategies in Algorithm 3. More specifically, Algorithm 3 first initializes a priority queue q and sets the priority of every edge by $+\infty$ (line 2–5). Then, the algorithm works in k rounds. At each round, the algorithm first gets the top element e from the priority queue (line 10) and computes $d(e)$ for e in $G = (V, E \backslash S)$ (line 11). The upper bound pruning strategy is implemented in line 12–14. If $d(e)$ is smaller than the current maximal marginal gain ($\Delta_{\max}$), the algorithm updates the priority of edge e by $d(e)$ and adds it back to the priority queue (line12–14). This pruning strategy can significantly reduce the computational cost in the first round, because we do not need to compute Δ_e for every edge e. If $d(e) > \Delta_{\max}$, then the algorithm recomputes Δ_e for

Algorithm 3 Accelerated greedy edge removal algorithm

Input: Graph $G = (V, E)$ and k
Output: A set S with k edges

```
1: S ← ∅
2: Initialize a priority queue q ← ∅
3: for each edge e ∈ E do
4:    Let p_e be the priority of edge e and set p_e ← +∞
5:    Add e as well as its priority into q
6: for i = 1 to k do
7:    Δ_max ← −∞
8:    e_max ← NULL
9:    while Δ_max < q.topPriority() do
10:      e ← q.pop()
11:      Compute d(e) in G = (V, E\S)
12:      if d_e <= Δ_max then
13:         Set p_e ← d(e)
14:         Add e as well as its priority into q
15:      else
16:         Compute Δ_e in G = (V, E\S)
17:         Set p_e ← Δ_e
18:         Add e as well as its priority into q
19:         if Δ_max < Δ_e then
20:            Δ_max ← Δ_e
21:            e_max ← e
22:   S ← S ⋃ {e_max}
23:   Remove e_max from q
24:   Delete e_max from E
25: return S
```

the top element e and updates its priority (line 16–18). And then the algorithm records the current maximal marginal gain ($\Delta_{\max}$) as well as its corresponding edge ($e_{\max}$, line 19–21). Based on the CELF optimization, if the current maximal marginal gain is greater than or equal to the maximal priority of the element in the queue, then we are done. Finally, the algorithm adds the edge $e_{\max}$ into the answer set S and removes $e_{\max}$ from q and E (line 22–24). It is worth mentioning that in line 13, setting the priority of edge e to $d(e)$ does not affect the final result, because the algorithm will recompute the true priority when it is needed (line 16–17). The worse case time complexity of Algorithm 3 is at most $O(km^{1.5})$ by assuming that the algorithm needs to evaluate Δ_e for all the edges in $G = (V, E\backslash S)$. However, in our experiments, Algorithm 3 significantly reduces the running time of Algorithm 2 by several orders of magnitude. Also, we can easily derive that the space complexity of Algorithm 2 is $O(m+n)$.

3.5 Approximate marginal gain evaluation

Recall that in Algorithm 3, the most time-consuming step is to compute the marginal gain Δ_e (line 16). Here, we propose a more efficient greedy algorithm with approximate marginal gain evaluation using the well-known FM sketch [16]. We refer to this new greedy algorithm as the approximate greedy edge removal algorithm.

The FM sketch is a probabilistic counting data structure, which can be used to estimate the cardinality of a multi-set efficiently [16]. Denote by N the cardinality of a multi-set A. The FM sketch is able to estimate N accurately using only $\log N + r$ bits, where r is a small constant. More specifically, the FM sketch is a bitmap with size $l = \log N + r$. There is a hash function $h : A \to \{1, \ldots, l\}$, which maps an element a ($a \in A$) to a bit i ($i \in \{1, \ldots, l\}$) in

the bitmap with probability $\Pr(h(a) = i) = 1/(2^{i+1})$. First of all, all the bits in the bitmap are initialized to 0. To process an element a ($a \in A$), we set the corresponding $h(a)$th bit of the bitmap to 1. Then, it has turned out that $2^z/0.77351$ is an asymptotically unbiased estimator of the cardinality N where z is the position of the least-significant zero bit in the bitmap. An important property of the FM sketch is that it can also be applied to estimate the cardinality of the union of two multi-sets if these multi-sets come from the same domain. In particular, we create two FM sketches with the same size for two multi-sets, respectively. To estimate the cardinality of the union of two multi-sets, we can perform a bitwise-OR between the two FM sketches and then estimate the cardinality based on the resulting FM sketch. To improve the estimation accuracy, we can utilize multiple hash functions. We remark that there are many other probabilistic counting structures, such as Loglog sketch [13] and Hyper Loglog sketch [15]. Here, we choose FM sketch because it is easy to implement and its performance is very good as indicated in the experiments.

The basic idea of our algorithm is described as follows. Initially, for each node v, we create an FM sketch $FM(v)$ to sketch the set of all the neighbor nodes of v denoted by $N(v)$. Then, in each round, we estimate $\Delta_e = d_u + d_v - |N(u) \cup N(v)|$ for each edge $e = (u, v)$ using the FM sketches $FM(u)$ and $FM(v)$. In particular, we first perform a bitwise-OR operation between $FM(u)$ and $FM(v)$ and then we estimate $|N(u) \cup N(v)|$ using the resulting FM sketch. Denote by $e_{\max} = (u_{\max}, v_{\max})$ the edge with maximal estimated marginal gain. Then, we add $e_{\max}$ into the answer set S. After that, we rebuild the FM sketches for the nodes $u_{\max}$ and $v_{\max}$. Note that a bitwise-OR operation can be done in constant time [39]. Thus, each marginal gain evaluation can be done in constant time using FM sketches. In addition, we can also use the pruning techniques presented in the previous subsection. The detailed description of the approximate greedy edge removal algorithm is very similar to Algorithm 3, thus we omit it for brevity. Theoretically, by a similar analysis presented in [22], the approximate greedy edge removal algorithm can achieve a $1 - 1/\mathbf{e} - \epsilon$ approximation factor with high probability by setting an appropriate number of hash functions, where ϵ is a small constant.

We analyze the time and space complexity of the approximate greedy algorithm as follows. First, the algorithm has to traverse the graph one time to create the FM sketches for all the nodes. Clearly, this procedure takes $O(m + n)$ time complexity. Second, in each round, the algorithm needs to rebuild the FM sketches for the end nodes of the selected edge. Clearly, the time overhead of this step is dominated by $O(m)$. Finally, in the worse case, the algorithm needs to evaluate the marginal gains for every edge in each round. Since each marginal gain computation takes constant time using FM sketches, the time complexity in each round is $O(m)$. There are k rounds in total. Thus, the time complexity of the approximate greedy edge removal algorithm is $O(km)$, which is linear with respect to the graph size. For the space complexity, the algorithm has to store the graph using $O(m+n)$ space. Each FM sketch takes $O(\log n)$ bits, which is equivalent to $O(1)$ words. Therefore, the total space overhead of the FM sketches is $O(n)$. Put it all together, the space complexity of the approximate greedy edge removal algorithm is $O(m + n)$.

4 Algorithms for node removals

In this section, we develop three algorithms for Problem 2, i.e., Eq. (3). Similar to the algorithms for Problem 1, first, we propose a degree-based node removal algorithm, which is without performance guarantee. Then, we present a greedy node removal algorithm by exploiting the submodularity property of the objective function $J(X)$, which is shown to be near optimal. Also, we propose two pruning techniques to speed up the greedy algorithm.

Algorithm 4 Degree-based node removal algorithm

Input: Graph $G = (V, E)$ and k
Output: A set X with k nodes

1: Compute d_v for each node v
2: $X \leftarrow \emptyset$
3: **for** $i = 1$ to k **do**
4: Find $v_{\max} = \arg\max_{v \in (V \setminus X)} d_v$
5: $X \leftarrow X \bigcup \{v_{\max}\}$
6: **for** each neighbor u of node $v_{\max}$ **do**
7: $d_u \leftarrow d_u - 1$
8: Delete $v_{\max}$ as well as its incident edges from G
9: **return** X

4.1 Degree-based node removal algorithm

Similar to the degree-based edge removal algorithm, here we introduce a degree-based node removal algorithm. The algorithm iteratively deletes k nodes with maximal degree. We describe our algorithm in Algorithm 4. First, Algorithm 4 calculates the degree for every node in G (line 1). Second, the algorithm works in k rounds. At each round, the algorithm finds the node $v_{\max}$ with maximal degree (line 4) and adds it into the answer set X. Then, the algorithm updates the degree of the neighbors of $v_{\max}$ and deletes $v_{\max}$ as well as its incident edges from G (line 6–8). We can easily derive that the time and space complexity of Algorithm 4 is $O(kn + m)$ and $O(m + n)$, respectively.

4.2 Greedy node removal algorithm

Similar to the greedy edge removal algorithm for Problem 1, we propose a greedy node removal algorithm for Problem 2. Let Δ_v be the number of triangles that contain node v. The greedy node removal algorithm iteratively removes k nodes with maximal Δ_v. We depict our algorithm in Algorithm 5. Let G_X be the induced subgraph by the nodes in $V \setminus X$. The algorithm works in k rounds. At each round, the algorithm computes Δ_v for every node in G_X (line 6) and then finds the node $v_{\max}$ with maximal Δ_v (line 7–9). Then, the algorithm adds $v_{\max}$ into the answer set (line 10), and deletes $v_{\max}$ as well as its incident edges from G. The most time-consuming step in Algorithm 5 is line 6, which requires $O(m^{1.5})$ time. Since the algorithm works in k rounds, the time complexity of Algorithm 5 is $O(km^{1.5})$. Also, we can derive that the space complexity of Algorithm 5 is $O(m + n)$.

4.3 Justification

We justify our algorithms presented above. First, we give an explanation for Algorithm 4. Then, we prove that Algorithm 5 achieves a $1 - 1/\mathbf{e}$ approximate factor. Our interpretation for Algorithm 4 is based on the following theorem.

Theorem 4.1 $\Delta_v \leq d_v(d_v - 1)/2$.

Proof For every node v, the triangle contains a node v if and only if the triangle contains two different incident edges of node v. There are at most $d_v(d_v - 1)/2$ different combinations of two incident edges of v. Therefore, the number of triangles that contain node v, i.e., Δ_v, is bounded by $d_v(d_v - 1)/2$. □

Algorithm 5 Greedy node removal algorithm

Input: Graph $G = (V, E)$ and k
Output: A set X with k nodes

$X \leftarrow \emptyset$
for $i = 1$ to k **do**
  $v_{\max} \leftarrow NULL$
  $\Delta_{\max} \leftarrow -\infty$
  for each node v in $V \backslash X$ **do**
    Compute Δ_v in G_X
    if $\Delta_v > \Delta_{\max}$ **then**
      $\Delta_{\max} \leftarrow \Delta_v$
      $v_{\max} \leftarrow v$
  $X \leftarrow X \bigcup \{v_{\max}\}$
  Delete $v_{\max}$ as well as its incident edges from G
return X

According to Theorem 4.1, Algorithm 4 can be interpreted as removing node to the current graph that minimize the upper bound on the number of triangles of the resulting graph.

Second, we prove that Algorithm 5 is a $1 - 1/\mathbf{e}$ approximate algorithm by exploiting the submodularity of $J(X)$. Below, we first prove that $J(X)$ is a nondecreasing submodular function.

Theorem 4.2 *For any subset of nodes $X \subseteq V$, $J(X)$ is a nondecreasing submodular function with $J(\emptyset) = 0$.*

Proof Since $C_\emptyset$ is a zero $n \times n$ matrix, we have $J(\emptyset) = 0$. Let $X \subseteq Y \subseteq V$ be two subsets of nodes and $v \notin Y$ be a node in V. Further, we define $\sigma_v(X) = J(X \bigcup \{v\}) - J(X)$ as the marginal gain. Then, to prove the nondecreasing property of $J(X)$, we need to show $\sigma_v(Y) \geq 0$. By definition, we have

$$\begin{aligned}\sigma_v(Y) &= tr((A - C_Y)^3) - tr((A - C_{Y\cup\{v\}})^3) \\ &= tr((A - C_Y)^3) - tr((A - C_Y - C_v)^3) \\ &= tr(C_v((A - C_Y)^2 + (A - C_Y)(A - C_Y - C_v) \\ &\quad + (A - C_Y - C_v)^2)) \\ &\geq 0, \end{aligned} \tag{6}$$

where the last inequality holds due to C_v, $A - C_Y$, and $A - C_Y - C_v$ are nonnegative matrices. Similarly, we have $\sigma_v(X) = tr(C_v((A-C_X)^2+(A-C_X)(A-C_X-C_v)+(A-C_X-C_v)^2)) \geq 0$. To prove the submodularity property of $J(X)$, we have to show $\sigma_v(X) \geq \sigma_v(Y)$. By definition, we have

$$\begin{aligned}\sigma_v(X) - \sigma_v(Y) &= tr((A - C_X)^3) - tr((A - C_Y)^3) \\ &= tr((C_X - C_Y)((A - C_X)^2 + (A - C_X)(A - C_Y) \\ &\quad + (A - C_Y)^2)) \\ &\geq 0, \end{aligned}$$

where the last inequality holds because $C_X - C_Y$, $A - C_X$, and $A - C_Y$ are nonnegative matrices. This completes the proof. □

Based on Theorem 4.2, Problem 2 is a submodular function maximization subject to cardinality constraint problem. By a well-known result in [38], there is a near-optimal greedy

algorithm for solving Problem 2. Below, we prove that our Algorithm 5 is indeed such a greedy algorithm. Let $\Delta_v(X)$ be the number of triangles in G_X that contain v. Our goal is to prove that the marginal gain $\sigma_v(X)$ is proportional to $\Delta_v(X)$. The following theorem shows that $\sigma_v(X) = 6\Delta_v(X)$.

Theorem 4.3 $\sigma_v(X) = 6\Delta_v(X)$.

Proof Without loss of generality, we assume that $|X| = x$, $X = \{v_{n-x+1}, \ldots, v_n\}$, and $v = v_{n-x}$. Then, the matrices $A - C_X$ and $A - C_{X\cup\{v\}}$ can be represented as two block matrices as follows.

$$(A - C_X) = \begin{pmatrix} Z & U \\ U^T & 0 \end{pmatrix},$$

and

$$(A - C_{X\cup\{v\}}) = (A - C_X - C_v) = \begin{pmatrix} Z & 0 \\ 0 & 0 \end{pmatrix},$$

where $Z \in \mathbb{R}^{(n-x-1)\times(n-x-1)}$ is the adjacency matrix of the graph $G_{X\cup\{v\}}$ (the induced subgraph by the nodes in $V\backslash\{X \cup \{v\}\}$), U is a $(n - x - 1) \times (x + 1)$ matrix and

$$U = \begin{pmatrix} A(1, n-x) & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ A(n-x-1, n-x) & 0 & \cdots & 0 \end{pmatrix}. \tag{7}$$

Then, we have

$$C_v = \begin{pmatrix} Z & U \\ U^T & 0 \end{pmatrix} - \begin{pmatrix} Z & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & U \\ U^T & 0 \end{pmatrix},$$

$$(A - C_X)^2 = \begin{pmatrix} Z^2 & 0 \\ 0 & 0 \end{pmatrix},$$

$$(A - C_X)(A - C_X - C_v) = \begin{pmatrix} Z^2 & ZU \\ 0 & 0 \end{pmatrix},$$

and

$$(A - C_X - C_v)^2 = \begin{pmatrix} Z^2 + UU^T & ZU \\ U^T Z & U^T U \end{pmatrix}.$$

Further, we have

$$C_v(A - C_X)^2 = \begin{pmatrix} 0 & 0 \\ U^T Z^2 & 0 \end{pmatrix},$$

$$C_v(A - C_X)(A - C_X - C_v) = \begin{pmatrix} 0 & 0 \\ U^T Z^2 & U^T ZU \end{pmatrix},$$

and

$$C_v(A - C_X - C_v)^2 = \begin{pmatrix} UU^T Z & UU^T U \\ U^T Z^2 + U^T UU^T & U^T ZU \end{pmatrix}.$$

By Eq. (6), we have

$$\begin{aligned}\sigma_v(X) &= tr(C_v(A - C_X)^2) + tr(C_v(A - C_X)(A - C_X - C_v)) \\ &\quad + tr(C_v(A - C_X - C_v)^2)) \\ &= 0 + tr(U^T ZU) + tr(UU^T Z) + tr(U^T ZU) \\ &= 3tr(U^T ZU). \end{aligned} \tag{8}$$

By substituting Z and U [Eq. (7)] into Eq. (8), we can get

$$\begin{aligned}\sigma_v(X) &= 3 \sum_{i=1}^{n-x-1} \sum_{j=1}^{n-x-1} A(i, j)A(i, n - x)A(j, n - x) \\ &= 6\Delta_{v_{n-x}}(X). \end{aligned}$$

This completes the proof. □

Based on Theorem 4.3, Algorithm 5 is a greedy algorithm that iteratively removes k nodes with maximal marginal gain. By a well-known result in [38], Algorithm 5 achieves $1 - 1/\mathbf{e}$ approximation factor as stated in the following theorem.

Theorem 4.4 *Algorithm 5 is a* $1 - 1/\boldsymbol{e}$ *approximation algorithm for Problem 2.*

4.4 Pruning techniques

In Algorithm 5, at each round, the most time-consuming step is line 6, which requires to compute Δ_v for each node v in G_X. Like Algorithm 3, here we present two similar pruning techniques to accelerate Algorithm 5 without loss of approximation factor. The first pruning technique is based on the submodularity property of the objective function $J(X)$ (CELF optimization) [28,37], and the second pruning technique is based on Theorem 4.1 (degree-based pruning). We refer to Algorithm 5 with these two pruning techniques as the accelerated greedy node removal algorithm. We outline our algorithm in Algorithm 6.

Like Algorithm 3, Algorithm 6 also makes use of a priority queue to find the node with maximal marginal gain. First, the algorithm initializes the answer set X to an empty set (line 1), and adds all the nodes with priority $+\infty$ into the priority queue (line 2–5). Then, the algorithm works in k rounds to find k nodes. At each round, the algorithm first initializes the current maximal marginal gain $\Delta_{\max}$ to $-\infty$ and the corresponding node $v_{\max}$ to null (line 7–8). Then, the algorithm accesses the top element v of the priority queue (line 10) and computes d_v for node v w.r.t. graph G_X. If $d_v \times (d_v - 1)/2 \leq \Delta_{\max}$, we have $\Delta_v \leq \Delta_{\max}$ by Theorem 4.1. Hence, the algorithm does not compute Δ_v. Instead, the algorithm updates the priority of node v by $d_v \times (d_v - 1)/2$ (line 12–14). If $d_v \times (d_v - 1)/2 > \Delta_{\max}$, the algorithm computes Δ_v for v in graph G_X and then updates the priority of node v by the newly computed Δ_v (line 16–18). If $\Delta_v \geq \Delta_{\max}$, the algorithm updates $\Delta_{\max}$ and $v_{\max}$ by Δ_v and v, respectively (line 19–21). Finally, the algorithm adds $v_{\max}$ into the answer set X and deletes $v_{\max}$ from the priority queue and the graph G. Note that the worse case time complexity of Algorithm 6 is at most $O(km^{1.5})$ by assuming that Δ_v for every node needs to evaluate at each round. In practice, we show that Algorithm 6 is faster than Algorithm 5 by several orders of magnitude. It is easy to derive that the space complexity of Algorithm 6 is still $O(m + n)$.

Algorithm 6 Accelerated greedy node removal algorithm

Input: Graph $G = (V, E)$ and k
Output: A set X with k nodes

```
1: X ← ∅
2: Initialize a priority queue q ← ∅
3: for each node v ∈ V do
4:    Let p_v be the priority of node v and set p_v ← +∞
5:    Add v as well as its priority into q
6: for i = 1 to k do
7:    Δ_max ← −∞
8:    v_max ← NULL
9:    while Δ_max < q.topPriority() do
10:      v ← q.pop()
11:      Compute d_v in G_X
12:      if d_v × (d_v − 1)/2 ≤ Δ_max then
13:         Set p_v ← d_v × (d_v − 1)/2
14:         Add v as well as its priority into q
15:      else
16:         Compute Δ_v in G_X
17:         Set p_v ← Δ_v
18:         Add v as well as its priority into q
19:         if Δ_max < Δ_v then
20:            Δ_max ← Δ_v
21:            v_max ← v
22:   X ← X ⋃ {v_max}
23:   Remove v_max from q
24:   Delete v_max as well as its incident edges from G
25: return X
```

4.5 Approximate marginal gain computation

Similar to the approximate greedy edge removal algorithm, here we present an approximate greedy node removal algorithm where the marginal gains Δ_v for a node v is estimated using FM sketch. Specifically, for each node v, we create an FM sketch $FM(v)$ to sketch the neighbor-set $N(v)$. Then, in each round, we estimate Δ_v using the formula $\Delta_v = \sum_{e=(u,v)\in E} \Delta_e/2$, where Δ_e can be estimated by the FM sketches $FM(u)$ and $FM(v)$. After selecting the node $v_{\max}$ with maximal estimated marginal gain, we rebuild the FM sketches for all the nodes in $N(v_{\max})$. Also, we can use the pruning techniques developed in the previous subsection to further speed up the algorithm. The detailed description of this algorithm is similar to Algorithm 6, thus we omit it for brevity.

Below, we show that the time and space complexity of the approximate greedy node removal algorithm is linear with respect to the graph size. First, it is easy to show that the total time cost for FM sketches construction and reconstruction is $O(m + n)$. Second, in the worse case, the algorithm has to evaluate the marginal gains for all nodes in each round. Clearly, the time complexity to estimate the marginal gain for a node v is $O(d_v)$ using FM sketch. Therefore, in each round, the worse case time complexity of marginal gain evaluation is $O(\sum_{v\in V} d_v) = O(m)$. Since there are k rounds in total, the time complexity of the algorithm is $O(km)$. For the space complexity, the algorithm needs to maintain the graph G and n FM sketches, which takes $O(m + n)$ space.

5 Experiments

Different algorithms We compare the proposed algorithms with four baseline algorithms. (1) *Random*: this algorithm removes k edges (nodes) randomly. (2) *PageRank*: it removes k edges (nodes) based on the PageRank scores [7]. Specifically, let p_u be the PageRank score of node u and $p_e = \min(p_u, p_v)$ be the PageRank score of an edge e. Then, for Problem 1, *PageRank* removes k edges with the highest p_e for $e \in E$. For Problem 2, *PageRank* deletes k nodes with the highest p_u for $u \in V$. (3) *NetMelt* [45]: It removes k edges based on the leading eigenvector of the adjacency matrix of the graph. (4) *NetShield* [46]: similar to *NetMelt*, it deletes k nodes based on the leading eigenvector. Detailed descriptions of *NetMelt* and *NetShield* can be found in [45] and [46], respectively.

We implement five proposed algorithms which are: (1) *Degree*, i.e., the degree-based edge (node) removal algorithm (Algorithms 1 and 4), (2) *Greedy*, i.e., the greedy edge (node) removal algorithm (Algorithms 2 and 5), (3) *GreedyCELF*, i.e., the greedy algorithms with CELF optimization, (4) *GreedyAll*, the greedy algorithms with both CELF optimization and degree-based pruning (Algorithms 3 and 6), and (5) *Approx*, i.e., the greedy algorithms with approximate marginal gain evaluation. In all the experiments, we set the number of hashing functions in *Approx* denoted by R to 100 (i.e., $R = 100$) because $R = 100$ is sufficient to guarantee a very good performance.

Evaluation methodology We compare the number of triangles in the resulting graph after k edges (nodes) removal to evaluate the effectiveness of various algorithms. In the resulting graph, the less the number of triangles is the more effective the algorithm is. We use the running time, which is measured by wall-clock time, to evaluate the efficiency of different algorithms.

Datasets We use 12 real-world datasets in the experiments. The datasets include two co-authorship networks (Astroph [27] and DBLP), five online social networks (Epinions [27], Douban, Delicious, Digg, and Flickr [51]), one location-based social network (Gowalla [27]), and two communication networks (EmailEuAll and WikiTalk [27]), one road network (roadNetCA [27]), and one web graph (webGoogle [27]). For all the datasets, if the graph is a directed graph, we ignore the direction of the edges in the graph. The detailed statistic information of the datasets is described in Table 1.

Experimental environment: We implement all the algorithms in C++. All of our experiments are conducted on a Windows Server 2007 with 4xDual-Core Intel Xeon 2.66 GHz CPU and 8G memory.

Table 1 Summary of the datasets

Name	# of nodes	# of edges	$d_{\max}$	$\bar{d}$	Power-law exponent α
Astroph	18,772	396,100	504	21.1	2.6
DBLP	78,649	382,294	756	4.9	2.4
Epinions	75,872	396,026	3,622	5.2	1.7
Douban	154,908	654,324	1,021	4.2	2.0
Delicious	537,392	1,459,778	5,615	2.7	1.6
Digg	771,231	6,044,952	310	7.8	3.3
Flickr	80,513	11,799,764	6,108	146.6	2.1
Gowalla	196,591	1,900,654	14,730	9.7	2.7
EmailEuAll	265,182	224,372	85	0.8	1.3
WikiTalk	2,394,381	8,235,230	1,089	3.5	1.7
roadNetCA	1,965,206	5,533,214	12	2.8	7.0
webGoogle	875,708	3,058,376	6,353	11.7	2.9

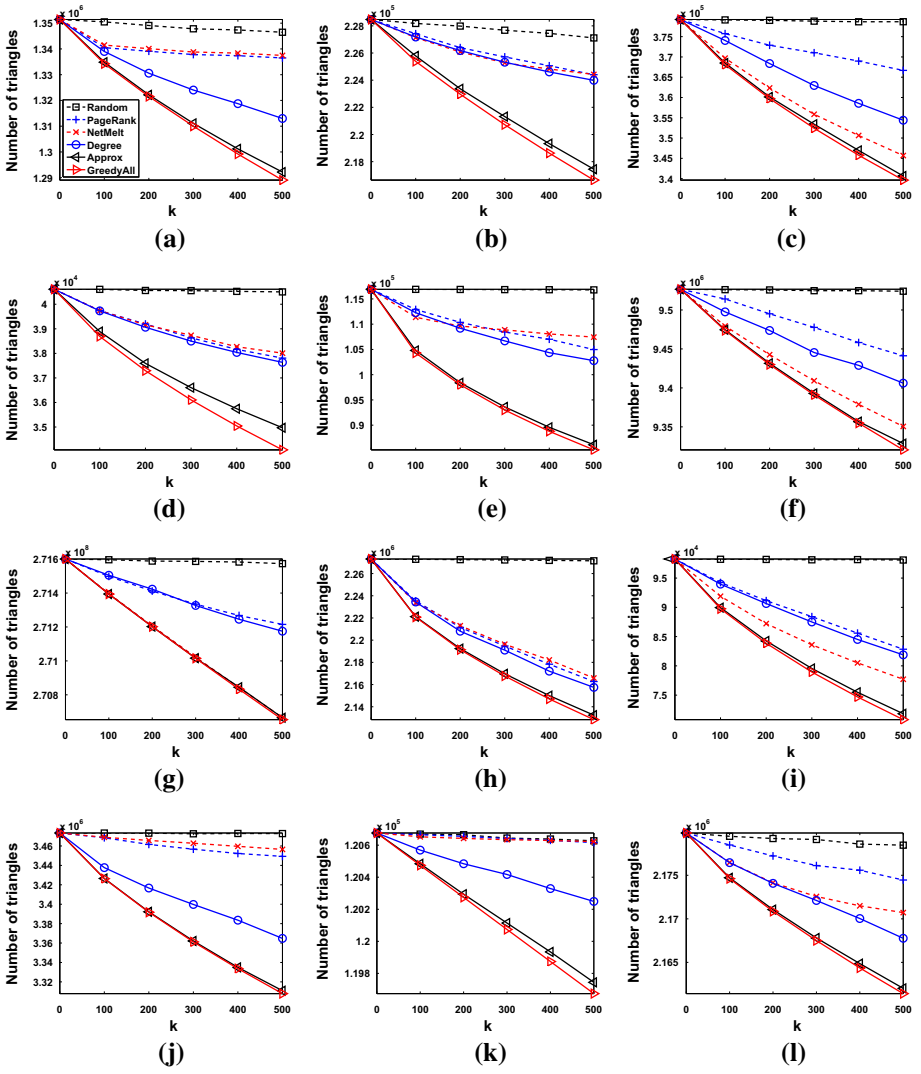


Fig. 1 Comparison of different edge removal algorithms. (number of triangles vs. k edges removal) **a** AstroPh, **b** DBLP, **c** Epinoinos, **d** Douban, **e** Delicious, **f** Digg, **g** Flickr, **h** Gowalla, **i** EmailEuAll, **j** WikiTalk, **k** roadNetCA, **l** webGoogle

5.1 Experimental results

Effectiveness of different algorithms Figures 1 and 2 depict the result of different edge removal algorithms and node removal algorithms over 12 datasets, respectively. For the greedy edge (node) removal algorithm, we only report the results of *GreedyAll* and *Approx* in Fig. 1 (Fig. 2), because both *GreedyCELF* and *Greedy* do not reduce the effectiveness of *GreedyAll*. From Fig. 1, we can find that *GreedyAll* consistently outperforms the other algorithms over all 12 datasets given k ranging from 0 to 500. The performance of *Approx* is very close to that of *GreedyAll*, which confirms the theoretical analysis in Sects. 3.5 and 4.5.

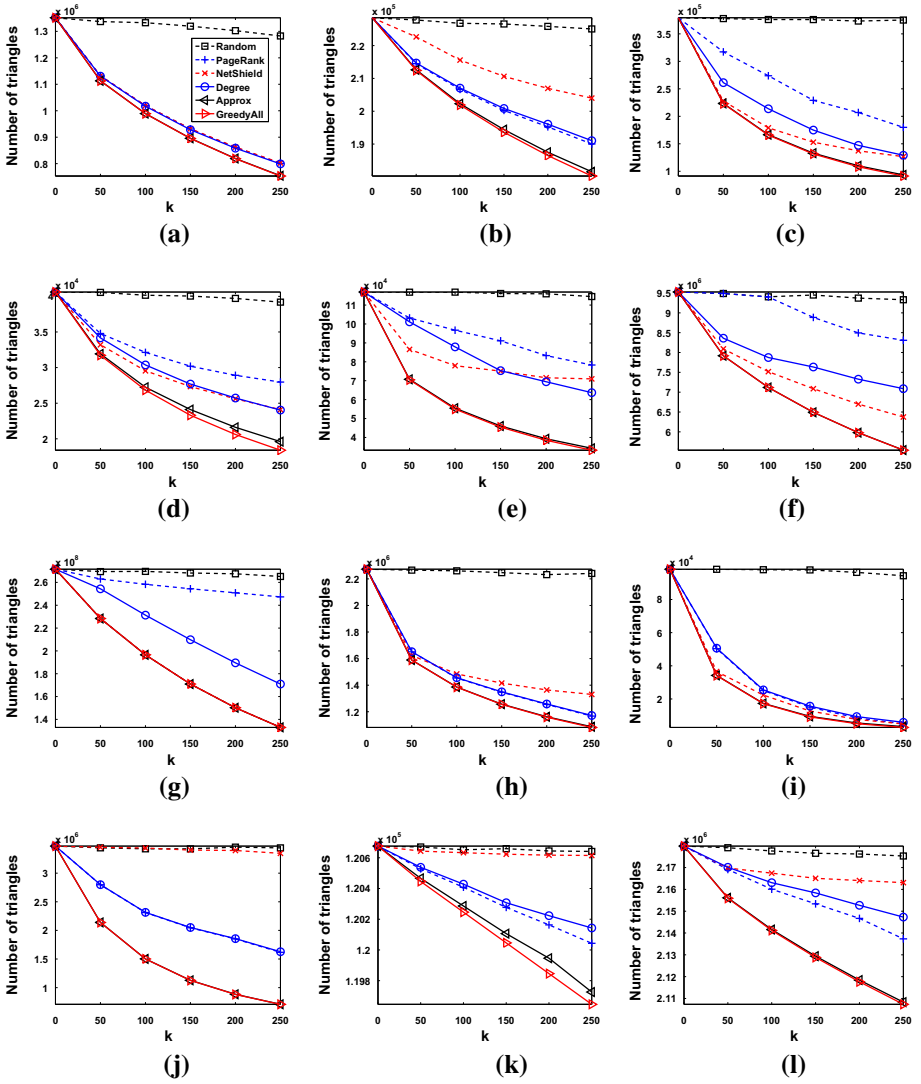


Fig. 2 Comparison of different node removal algorithms (number of triangles vs. k nodes removal) **a** AstroPh, **b** DBLP, **c** Epinoinos, **d** Douban, **e** Delicious, **f** Digg, **g** Flickr, **h** Gowalla, **i** EmailEuAll, **j** WikiTalk, **k** roadNetCA, **l** webGoogle

Moreover, in most datasets, the gap between the curves of *GreedyAll* and *Approx* (red and black solid curves) and the curves of the other algorithms increases as k increases. This is because *GreedyAll* and *Approx* are the near-optimal approximation algorithms, which achieve $1 - 1/\mathbf{e}$ and $1 - 1/\mathbf{e} - \epsilon$ approximation factors, respectively. Furthermore, such approximation factors are independent of k. Therefore, when k increases, these algorithms can still achieve a very good performance. Instead, *Degree* (Algorithm 1) and the baseline algorithms are without any performance guarantee, thereby the performance could decrease as k increases. In addition, we can find that *Degree* outperforms the baseline algorithms in most datasets. This could be because *Degree* selects the edge with highest *degree* ($d(e)$

Table 2 Running time of different edge removal algorithms

Time (s)	*Random*	*PageRank*	*NetMelt*	*Degree*	*Greedy*	*GreedyCELF*	*GreedyAll*	*Approx*
Astroph	0.014	0.531	0.732	0.084	434.5	5.854	1.285	1.081
DBLP	0.014	0.942	2.031	0.204	80.8	1.857	1.105	1.095
Epinions	0.017	1.987	2.531	1.723	574.9	6.956	2.289	2.220
Douban	0.016	3.005	3.789	1.979	292.9	4.565	3.493	3.389
Delicious	0.013	4.329	8.091	0.418	707.4	10.9	4.461	4.521
Digg	0.012	21.12	23.28	0.662	1,9361	215.3	30.09	25.38
Flickr	0.015	16.89	20.99	0.289	166,148	1813	61.17	35.12
Gowalla	0.014	3.784	4.558	0.645	10,097	107.5	7.036	5.324
EmailEuAll	0.013	1.189	1.248	0.861	166.1	2.272	1.252	1.23
WikiTalk	0.013	49.44	52.48	4.908	38,577	409.7	99.03	63.29
roadNetCA	0.013	14.23	17.56	2.356	321.1	24.17	17.89	16.21
webGoogle	0.015	10.89	14.73	3.360	482.8	13.68	8.413	8.217

defined in Sect. 3.1), which is an upper bound of the marginal gain. This result suggests that *Degree* is a very good heuristic algorithm for the triangle minimization problem. It also indicates that the degree can be a very effective pruning strategy in *GreedyAll*. We will illustrate this point in the following experiment. Also, we can find that *Random* performs poorly over all the datasets, as it does not take any graph-structural information into account.

Similarly, for the node removal algorithms, we can observe that *GreedyAll* (Algorithm 6) and *Approx* significantly outperform the other competitors over all the datasets from Fig. 2. Likewise, the results of *Approx* are very similar to the results of *GreedyAll* which further confirm that the approximate greedy algorithm is very effective. In addition, we can find that *Degree*, *PageRank* and *NetShield* achieve relatively good performance in certain datasets (e.g., in Astroph, Gowalla and EmailEuAll datasets), but in the remaining datasets they perform poorly. Moreover, in all the datasets, the gap between the curves of *GreedyAll* and *Approx* and the curves of the other algorithms increases with increasing k. This is because *Degree* and the baselines algorithms are without any performance guarantee. These results are consistent with the theoretical analysis presented in Sects. 3.3 and 4.3.

Efficiency testing and the power of pruning In this experiment, we show the efficiency of different algorithms and the effective of the pruning strategies in *GreedyAll*. Below, we only report the results of the edge removal algorithms, because similar results can be observed for the node removal algorithms. Table 2 depicts the running time of different edge removal algorithms given that $k = 100$. Similar results can be obtained for other k. As can be seen, *Random* is the most efficient algorithm, because it removes edges randomly. *GreedyAll* and *Approx* are also very efficient, which take around 99 and 63 s in the WikiTalk dataset (more than 2 million nodes and 8 million edges), respectively. In general, *Approx* is more efficient than *GreedyAll*. As desired, the running time of *Approx* is comparable to those of *PageRank* and *NetMelt* because all of them have linear time complexity. Among *Greedy*, *GreedyCELF*, and *GreedyAll*, the *GreedyAll* algorithm is clearly the winner as it integrates two pruning strategies. Now, we analyze the power of these two pruning strategies described in Sect. 3.4. Specifically, let us focus on the columns 6–8 in Table 2. *GreedyAll* reduces the running time of *Greedy* by 524.8 times on average over all the datasets, and *GreedyCELF* shortens the running time of *Greedy* by 68.4 times on average. These results imply that the CELF optimization is a very effective pruning rule, which consists with the previous observation in [28], and the degree-based pruning strategy is also very powerful in real-word graphs.

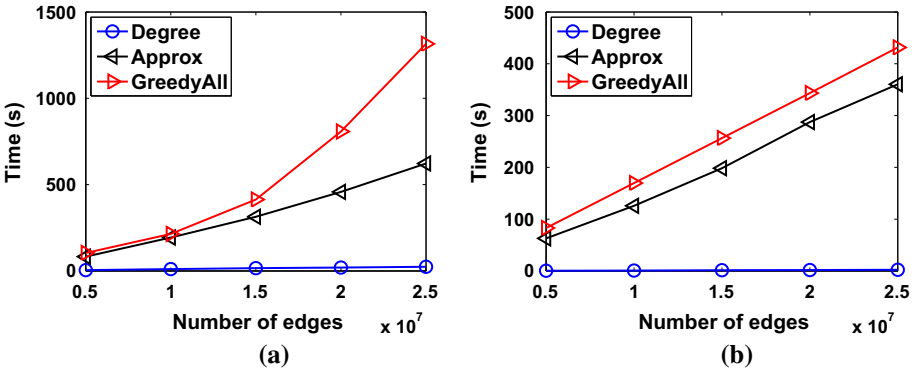


Fig. 3 Scalability testing **a** edge removal algorithms, **b** node removal algorithms

Scalability testing To test the scalability of our algorithms, we generate five large synthetic graphs based on a power-law random graph model [5]. Specifically, we generate five synthetic graphs $G_1, \ldots, G_5$ with G_i has i million nodes and $5 \times i$ million edges for $i = 1, \cdots, 5$. Figure 3a, b show the results of the edge removal algorithms and node removal algorithms, respectively, given that $k = 100$. Similar results can be observed for other k. For the greedy edge (node) removal algorithm, we only report the results of *GreedyAll* and *Approx* as they are faster than *GreedyCELF* and *Greedy*. From Fig. 3a, we can see that *Degree* (Algorithm 1) and *Approx* scale linear with respect to the number of edges because the complexity of *Degree* and *Approx* are linear. The curve of *GreedyAll* (Algorithm 3) is a concave-up and increasing curve, which consists with the complexity of the algorithm, i.e., $O(m^{1.5})$. Similarly, for the node removal algorithms, from Fig. 3b, we can observe that *Degree* (Algorithm 4) and *Approx* scale linearly with respect to the number of edges, because their time complexity are linear. Surprisingly, we find that *GreedyAll* (Algorithm 6) also scales linearly with respect to the number of edges. The reason could be that the pruning strategies (both CELF optimization and degree-based pruning) are very effective for the greedy node removal algorithm. Additionally, we remark that the linear scalability (w.r.t. the graph size) of *Approx* is independent of the number of hashing functions used (i.e., the parameter R). In fact, we have observed (not shown) that for a large R, the scalability of *Approx* is still linear with respect to the graph size, which is consistent with the time complexity of *Approx*.

Clustering coefficient versus k edges (nodes) removal Here, we examine the variation of clustering coefficient of a network when k edges (nodes) of the network are removed by different algorithms. For brevity, we only report the results observed in Astroph dataset, and very similar results can be observed from the other datasets. Also, for the greedy edges (nodes) removal algorithms, we only report the results of *GreedyAll* and *Approx*. Similar results can be observed from other datasets. Figure 4 depicts our results. For *GreedyAll* (both edge removal and node removal) and *Approx*, we can see that the clustering coefficient decreases near linearly as k increases. This result indicates that the greedy algorithms are very effective for decreasing the clustering coefficient of a network. In general, for *Degree* (both edge removal and node removal) and the other baseline algorithms, we can find that the clustering coefficient slightly decreases as k increases and the curves of these algorithms tend to be flat with increasing k. These results suggest that the clustering coefficient of a network is very robust to such algorithms.

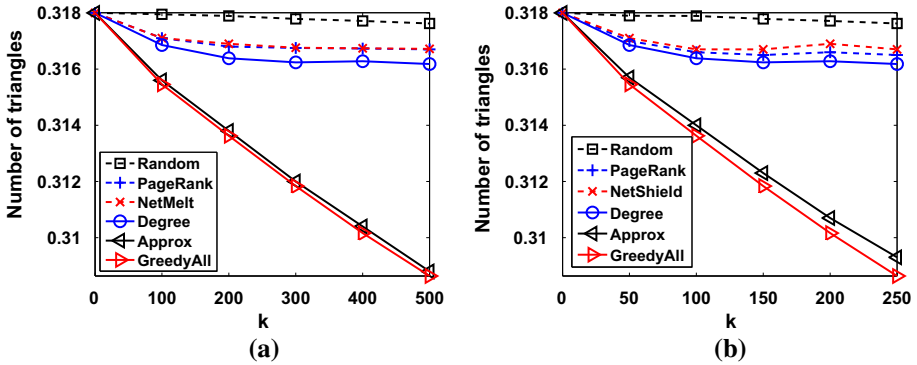


Fig. 4 Clustering coefficient versus k edges (nodes) removal in Astroph dataset **a** edges removal algorithms, **b** nodes removal algorithms

Discussions Here, we discuss the advantages and disadvantages of the proposed algorithms based on our theoretical and experimental results. First, compared to the state-of-the-art *PageRank* and *NetShield* algorithms, the proposed *Degree* algorithm is much faster than *PageRank* and *NetShield* as shown in Table 2. Moreover, in most datasets, the effectiveness of *Degree* is significantly better than that of *PageRank*, and it is comparable with *NetShield*. Second, by effectiveness, the proposed *GreedyAll* and *Approx* significantly outperform the other competitors. This is because both of them are near optimal in theory, whereas the other algorithms are without any performance guarantee. As a consequence, the performance gap between *GreedyAll* (or *Approx*) and the other competitors increases as k increases (Figs. 1, 2). The running time of *Approx* is lower than that of *GreedyAll*, but it is much higher than the proposed *Degree* algorithm. However, the scalability testing results (Fig. 3) show that *Approx* scales near linearly, indicating that *Approx* can also work on very large graphs. To summarize, in some real-world applications, when the running time is a crucial factor, we recommend the proposed *Degree* algorithm, as it not only runs faster than the state-of-the-art algorithms, but it also performs as good as the state-of-the-art algorithms. For the applications where the effectiveness of the algorithm is more crucial than the running time, we recommend the proposed *Approx* algorithm, because it is near optimal for any k, and it can also scale to handle very large graphs.

6 Related work

Triangle counting Triangle counting has been well studied in the literature. The crude triangle counting algorithm is to enumerates all possible triples of nodes, which results in a $O(n^3)$ time complexity. The first fast triangle counting algorithm is based on spanning-tree, which achieves $O(m^{1.5})$ time complexity [20]. Then, Alon et al. [2] proposed a $O(m^{1.41})$ triangle counting algorithm based on fast matrix multiplication. However, their algorithm requires $O(n^2)$ space, which is impractical in large graphs. To overcome this issue, Schank and Wagner [41] presented several more practical triangle counting algorithms with $O(m+n)$ space complexity based on node or edge iteration. Subsequently, Latapy proved that the time complexity of the algorithm presented in [41] is $O(m^{1.5})$, and also he suggested several improved algorithms for triangle counting. All the aforementioned algorithms are in-memory algorithms. For a good survey of these algorithms, we point out to [40]. For the graph that is

too big to fit into memory, Chu and Cheng [10] proposed an I/O-efficient triangle counting algorithm. Suri and Vassilvitskii [44] proposed a parallel counting algorithm using Map-Reduce framework. Besides exact triangle counting, another line of research is to count the number of triangles approximately. For example, Bar-Yossef et al. [4] proposed a streaming algorithm for estimating the number of triangles in large graphs using sketch techniques. Further improved streaming algorithms are presented in [8,21]. Becchetti et al. [6] proposed a semi-streaming algorithm to estimate the number of local triangles for a given node in large graphs. More recently, Tsourakakis et al. [47] presented a sampling-based method for triangle counting in large graphs. Their method first reduces the edges in the graph by sampling and then assigns a weight to each residual edge. Then, they estimate the number of triangles based on the resulting weighted graph. Avron [3] proposed an approximate triangle counting algorithm based on randomized trace estimators. Seshadhri et al. [43] presented an efficient approximate triangle counting algorithm based on wedge sampling. Their algorithm can also be used to estimate the number of directed triangles in a directed graphs.

Submodular function maximization Our work is also related to the submodular function maximization problem [38]. Generally, the problem of submodular function maximization subject to cardinality constraint is NP-hard. Nemhauser et al. [38] proposed a greedy algorithm with $1 - 1/\mathbf{e}$ approximate factor for solving this problem. Recently, many applications have been formulated as the submodular function maximization subject to cardinality constraint problem [14,22,23,28,30,34]. Examples include the maximal k coverage problem [14], the influence maximization problem in social networks [22], the outbreak detection problem in networks [28], the observation selection and sensor placement problem [23,25], the document summarization problem [34,35], the privacy preserving data publishing problem [24], the diversified ranking problem [29,30], and the random walk domination problem [31]. All of these problems can be approximately solved by the greedy algorithm given in [38]. Here, we study the triangle minimization problem, and we show that it can be formulated as a submodular function maximization problem. Also, we present a $1 - 1/\mathbf{e}$ greedy algorithm to solve it.

Network attack problems In the literature, there are several network attack problems related to our problems. We summarize these problems in Table 3. More specifically, Albert et al. [1] studied a network attack problem where the utility function is the diameter of a network. Subsequently, Schneider et al. [42] investigated a network attack problem where the utility function is the size of the MCC of a network. Both of these studies only focus on the robustness of a network under edge or node attacks. Both of them did not study the complexity of their problem and also no efficient algorithm was developed in these work. More recently, Li et al. [32] studied another network attack issue where the utility function is the size of the residual network. They shown that the problem is NP-hard, and they also proposed several efficient algorithm for solve their problems. However, the size of the residual network cannot capture the cohesiveness of a network. To achieve that end, in this work, we propose to use

Table 3 Summary of network attack problems

Utility function	Complexity	Efficient algorithm	Refs.
Diameter	Unknown	No	[1]
Size of MCC	Unknown	No	[42]
Size of residual graph	NP-hard	Yes	[32]
Number of triangles	NP-hard	Yes	This work

the number of triangles as an utility function, and we also devise several algorithms to solve our problems efficiently.

7 Conclusions

This paper presents a study of triangle minimization problem in large networks subject to a small fraction of edges (nodes) removal. In particular, under either edge removal or node removal, we formulate the triangle minimization problem as a submodular maximization problem. For the triangle minimization problem under edge (node) removal, we propose a degree-based edge (node) removal algorithm and a near-optimal greedy edge (node) removal algorithm. In addition, we further introduce two pruning strategies and a technique of approximate marginal gain computation to accelerate the greedy algorithms. Finally, we evaluate the proposed algorithms using 12 real-world graphs. The results show that our algorithms is very effective with respect to the baseline methods, and all of our algorithms scale very well in large graphs. Our idea of developing upper bound (Theorems 3.1, 4.1) for pruning in the greedy algorithm can also be used for other submodular maximization problem, such as the influence maximization problem [22]. It would be interesting to develop upper bound techniques to accelerate the greedy algorithm for such problems. In addition, the objective functions of Problems 1 and 2 are submodular. An interesting problem is to plus these two objective functions (or by a positive weight, it is still submodular) and then optimize both the edge attack and node attack simultaneously.

Acknowledgments We thank anonymous reviewers for their helpful comments. The work was supported in part by (1) NSFC Grant 61402292, Natural Science Foundation of SZU (Grant No. 201438) and (2) Research Grants Council of the Hong Kong SAR, China, 14209314 and 418512.

Appendix

Proof of Theorem 2.1 First, it is known that the set cover with frequency constraint (SCFC) problem is NP-hard [19,48]. Given a ground set $\mathcal{U}$, a collection of n subsets $\mathcal{S} = \{S_1, S_2, \cdots, S_n\}$ where $\bigcup_i S_i = \mathcal{U}$, and a frequency parameter t ($t < n$), the SCFC problem is to find the minimum number of subsets in $\mathcal{S}$ that covers all elements in $\mathcal{U}$. Here, the frequency parameter t denotes that every element in $\mathcal{U}$ is included in t subsets in $\mathcal{S}$. Let us consider a special case of the SCFC problem, which has an additional constraint that the intersection of any three subsets in $\mathcal{S}$ has at most one element (i.e., for any i, j, k and $i \neq j \neq k$, $|S_i \cap S_j \cap S_k| \leq 1$). For convenience, we refer to this problem as the intersection-bounded SCFC (IBSCFC) problem. Below, we show that the IBSCFC problem is also NP-hard. Suppose to the contrary that there is a polynomial algorithm $\mathcal{A}$ to solve the IBSCFC problem. For any $|S_i \cap S_j \cap S_k| > 1$ in the SCFC problem, we can discard the "redundant-common elements" in the subsets S_i, S_j, S_k so that $|\tilde{S}_i \cap \tilde{S}_j \cap \tilde{S}_k| = 1$ where $\tilde{S}_i$ denotes the subset S_i after discarding the redundant-common elements (i.e., for S_i, S_j, S_k, only one common element is left). Then, the SCFC problem becomes the IBSCFC problem, and we invoke algorithm $\mathcal{A}$ to solve it. It is important to note that the optimal solution (the selected subsets ID) obtained by algorithm $\mathcal{A}$ is the optimal solution for the SCFC problem. The reason is as follows. For any S_i, S_j, S_k with $|S_i \cap S_j \cap S_k| > 1$, the redundant-common elements are only in these three subsets (by our constraint, each element is included in three subsets), thus they do not affect the optimal solution. Moreover, the optimal solution obtained by algorithm $\mathcal{A}$

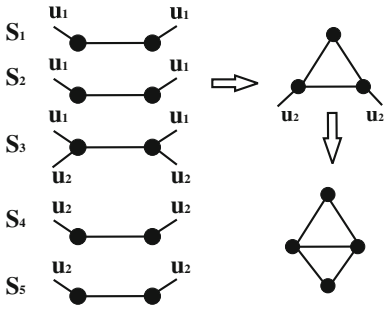


Fig. 5 Illustration of the graph construction

must contain at least one subset from S_i, S_j, S_k, because these three subsets have one common element left which must be covered by a subset in the optimal solution. By the above process, there is a polynomial algorithm for the SCFC problem, which is a contradiction.

Second, we consider the maximum coverage version of the IBSCFC problem, called IBMCFC, where the goal is to find k subsets in $\mathcal{S}$ to maximize the cardinality of their union. It is easy to show that this problem is also NP-hard. Because if not, there is a polynomial algorithm $\mathcal{B}$ to solve the IBMCFC problem. Since $\bigcup_i S_i = \mathcal{U}$, we can invoke $\mathcal{B}$ at most n times to get an optimal solution of the IBSCFC problem (enumerating k from 1 to n). That is to say, there is a polynomial algorithm for the IBSCFC problem, which is a contradiction.

Third, to prove the theorem, we show a reduction from the IBMCFC problem. Specifically, for each subset S_i, we create an edge e_i with $2|S_i|$ stubs, which are used to combine the end nodes of different edges. Each end node of an edge is associated with $|S_i|$ stubs, and these stubs are labeled by the element ID in S_i. Then, for any three subsets S_i, S_j, and S_k $(i \neq j \neq k)$ with $|S_i \cap S_j \cap S_k| = 1$, we combine the end nodes of their corresponding edges with the same stub labels so that they can form a triangle. As an example, let $\mathcal{U} = \{u_1, u_2\}$, $S_1 = \{u_1\}$, $S_2 = \{u_1\}$, $S_3 = \{u_1, u_2\}$, $S_4 = \{u_2\}$, $S_5 = \{u_2\}$. Clearly, each element in $\mathcal{U}$ is in three subsets and any three subsets have at most one common element. Then, for each subset, we create an edge with stubs as shown in the left part of Fig. 5. Then, we can construct a graph as shown in the right part of Fig. 5. By this construction, each triangle is represented by an element in $\mathcal{U}$, and each edge e_i in the resulting graph is represented by a subset S_i in $\mathcal{S}$. As a result, the optimal solution of the triangle minimization problem (by edge removal) in the resulting graph is the optimal solution of the IBMCFC problem. Since IBMCFC is NP-hard, the triangle minimization problem by edge removal is also NP-hard. This completes the proof. □

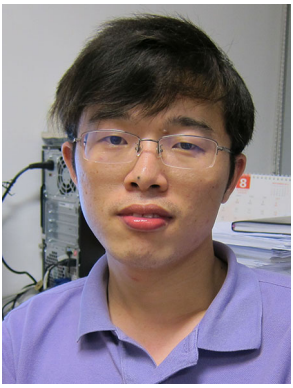
Proof of Theorem 2.2 Similar to the proof of Theorem 2.1, we can show a reduction from the IBMCFC problem. Following the notations used in the proof of Theorem 2.1, we create a graph G for the instance of triangle minimization problem by node removal as follows. Specifically, for each S_i in $\mathcal{S}$, we create a node v_i. For each pair S_i and S_j $(i \neq j)$, we create an edge (v_i, v_j) if and only if $S_i \bigcap S_j \neq \emptyset$. By this construction, each node is represented by a subset, and each triangle is represented by an element. One can easily check that the optimal solution of the triangle minimization problem by node removal is the optimal solution of the IBMCFC problem. Thus, the theorem is established.

References

1. Albert R et al (2000) Error and attack tolerance of complex networks. Nature 406:378–382

2. Alon N et al (1997) Finding and counting given length cycles. Algorithmica 17(3):209–223
3. Avron H (2010) Counting triangles in large graphs using randomized matrix trace estimation. In: Proceedings of KDD-LDMTA'10
4. Bar-Yossef Z et al (2002) Reductions in streaming algorithms, with an application to counting triangles in graphs. In: SODA
5. Barabasi A-L, Albert R (1999) Emergence of scaling in random networks. Science 286(5439):509–512
6. Becchetti L et al (2008) Efficient semi-streaming algorithms for local triangle counting in massive graphs. In: KDD
7. Brin S, Page L (1997) PageRank: bringing order to the web. Tech. rep, Stanford Digital Library Project
8. Buriol LS et al (2006) Counting triangles in data streams. In: PODS
9. Callaway DS et al (2000) Network robustness and fragility: percolation on random graphs. Phys Rev Lett 85(25):5468–5471
10. Chu S, Cheng J (2011) Triangle listing in massive networks and its applications. In: KDD
11. Cohen R et al (2000) Resilience of the internet to random breakdowns. Phys Rev Lett 85(21):5626–5628
12. Coleman JS (1988) Social capital in the creation of human capital. Am J Sociol 94:95–120
13. Durand M, Flajolet P (2003) Loglog counting of large cardinalities (extended abstract). In: ESA, pp 605–617
14. Feige U (1998) A threshold of in n for approximating set cover. J ACM 45(4):634–652
15. Flajolet P et al (2003) Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In: ESA, pp 605–617
16. Flajolet P, Martin GN (1985) Probabilistic counting algorithms for data base applications. J Comput Syst Sci 31(2):182–209
17. Godsil C, Royle GF (2001) Algebraic graph theory. Springer, Berlin
18. Hanneman RA, Riddle M (2005) Introduction to social network methods. University of California, Riverside. http://faculty.ucr.edu/~hanneman/nettext/
19. Hochbaum DS (1996) Approximation algorithms for NP-hard problems. PWS Publishing Company, Boston, MA
20. Itai A, Rodeh M (1978) Finding a minimum circuit in a graph. SIAM J Comput 7(4):413–423
21. Jowhari H, Ghodsi M (2005) New streaming algorithms for counting triangles in graphs. In: COCOON
22. Kempe D et al (2003) Maximizing the spread of influence through a social network. In: KDD
23. Krause A, Guestrin C (2007) Near-optimal observation selection using submodular functions. In: AAAI
24. Krause A, Horvitz E (2008) A utility-theoretic approach to privacy and personalization. In: AAAI
25. Krause A et al (2008) Near-optimal sensor placements in Gaussian processes: theory, efficient algorithms and empirical studies. J Mach Learn Res 9:235–284
26. Latapy M (2008) Main-memory triangle computations for very large (sparse (power-law)) graphs. Theor Comput Sci 407:1–3
27. Leskovec J (2010) Standford network analysis project
28. Leskovec J et al (2007) Cost-effective outbreak detection in networks. In: KDD
29. Li R-H, Yu JX (2011) Scalable diversified ranking on large graphs. In: ICDM
30. Li R-H, Yu JX (2013) Scalable diversified ranking on large graphs. IEEE Trans Knowl Data Eng 25(9):2133–2146
31. Li R-H et al (2014a) Random-walk domination in large graphs. In: ICDE
32. Li R-H et al (2012) Measuring robustness of complex networks under MVC attack. In: CIKM
33. Li R-H et al (2014b) Measuring the impact of MVC attack in large complex networks. Inf Sci 278:685–702
34. Lin H, Bilmes J (2010) Multi-document summarization via budgeted maximization of submodular functions. In: HLT-NAACL
35. Lin H, Bilmes J (2011) A class of submodular functions for document summarization. In: ACL
36. McPherson M et al (2001) Birds of a feather: homophily in social networks. Annu Rev Sociol 27:415–444
37. Minoux M (1978) Accelerated greedy algorithms for maximizing submodular set functions. Lecture Notes in Control and Information Sciences. Springer, Berlin
38. Nemhauser GL et al (1978) An analysis of approximations for maximizing submodular set functions-I. Math Program 14:265–294
39. Palmer CR et al (2002) ANF: a fast and scalable tool for data mining in massive graphs. In: KDD, pp 81–90
40. Schank T (2007) Algorithmic aspects of triangle-based network analysis. PhD Thesis, University Karlsruhe (TH)
41. Schank T, Wagner D (2005) Finding, counting and listing all triangles in large graphs, an experimental study. In: WEA
42. Schneider CM et al (2011) Mitigation of malicious attacks on networks. PNAS 108(10):3838–3841
43. Seshadhri C et al (2012) Fast triangle counting through wedge sampling. CoRR abs/1202.5230

44. Suri S, Vassilvitskii S (2011) Counting triangles and the curse of the last reducer. In: WWW
45. Tong H et al (2012) Gelling, and melting, large graphs by edge manipulation. In: CIKM
46. Tong H et al (2010) On the vulnerability of large graphs. In: ICDM
47. Tsourakakis CE et al (2009) DOULION: counting triangles in massive graphs with a coin. In: KDD
48. Vazirani VV (2001) Approximation algorithms. Springer, Berlin
49. Vondrak J (2010) Submodularity and curvature: the optimal algorithm. RIMS Kokyuroku Bessatsu B23:253–266
50. Watts DJ, Strogatz SH (1998) Collective dynamics of 'small-world' networks. Nature 393:440–442
51. Zafarani R, Liu H (2009) Social Computing Data Repository at ASU



Rong-Hua Li received the Ph.D. degree from the Chinese University of Hong Kong in 2013. He is currently an Assistant Professor at Shenzhen University, China. His research interests include algorithm design for (big) graph, matrix, and sequence data with applications in social network analysis, database and data mining. He is a member of the IEEE.

Jeffrey Xu Yu received the BE, ME, and the Ph.D. degrees in computer science, from the University of Tsukuba, Japan, in 1985, 1987, and 1990, respectively. He is currently a professor at the Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong. His major research interests include graph mining, graph database, keyword search, social network, and query processing and optimization. He is a senior member of the IEEE, a member of the IEEE Computer Society, and a member of ACM.