The K-clique Densest Subgraph Problem

Charalampos E. Tsourakakis
Harvard School of Engineering and Applied Sciences
babis@seas.harvard.edu

ABSTRACT

Numerous graph mining applications rely on detecting subgraphs which are large near-cliques. Since formulations that are geared towards finding large near-cliques are **NP**-hard and frequently inapproximable due to connections with the Maximum Clique problem, the poly-time solvable *densest subgraph problem* which maximizes the average degree over all possible subgraphs "lies at the core of large scale data mining" [10]. However, frequently the *densest subgraph problem* fails in detecting large near-cliques in networks.

In this work, we introduce the *k-clique densest subgraph problem*, $k \geq 2$. This generalizes the well studied *densest subgraph problem* which is obtained as a special case for $k = 2$. For $k = 3$ we obtain a novel formulation which we refer to as the *triangle densest subgraph problem*: given a graph $G(V, E)$, find a subset of vertices S^* such that $\tau(S^*) = \max_{S \subseteq V} \frac{t(S)}{|S|}$, where $t(S)$ is the number of triangles induced by the set S.

On the theory side, we prove that for any k constant, there exist an exact polynomial time algorithm for the *k-clique densest subgraph problem*. Furthermore, we propose an efficient $\frac{1}{k}$-approximation algorithm which generalizes the greedy peeling algorithm of Asahiro and Charikar [8, 18] for $k = 2$. Finally, we show how to implement efficiently this peeling framework on MapReduce for any $k \geq 3$, generalizing the work of Bahmani, Kumar and Vassilvitskii for the case $k = 2$ [10]. On the empirical side, our two main findings are that (i) the *triangle densest subgraph* is consistently closer to being a large near-clique compared to the *densest subgraph* and (ii) the peeling approximation algorithms for both $k = 2$ and $k = 3$ achieve on real-world networks approximation ratios closer to 1 rather than the pessimistic $\frac{1}{k}$ guarantee. An interesting consequence of our work is that *triangle counting*, a well-studied computational problem in the context of social network analysis can be used to detect large near-cliques. Finally, we evaluate our proposed method on a popular graph mining application.

WWW 2015, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3469-3/15/05.
http://dx.doi.org/10.1145/2736277.2741128 .

Categories and Subject Descriptors

G.2.2 [**Graph Theory**]: Graph Algorithms; H.2.8 [**Database Applications**]: Data mining

General Terms

Theory, Experimentation

Keywords

Densest subgraph problem; Graph algorithms; Graph Mining; Near-clique extraction

1. INTRODUCTION

A wide variety of data mining applications relies on extracting dense subgraphs from large graphs. In bioinformatics dense subgraphs are used for detecting protein complexes in protein-protein interaction networks [9] and for finding regulatory motifs in DNA [24]. They are also used for detecting link spam in Web graphs [26], graph compression [17] and mining micro-blogging streams [6].

Among the various formulations for finding dense subgraphs, the *densest subgraph problem* (DS-Problem) stands out for the facts that is solvable in polynomial time [28] and $\frac{1}{2}$-approximable in linear time [18, 40]. To state the DS-Problem we first introduce the necessary notation. In this work we focus on simple unweighted, undirected graphs. Given a graph $G = (V, E)$ and a subset of vertices $S \subseteq V$, let $G(S) = (S, E(S))$ be the subgraph induced by S, and let $e(S) = |E(S)|$ be the size of $E(S)$. Also, the *edge density* of the set S is defined as $f_e(S) = e(S)/\binom{|S|}{2}$. The DS-Problem maximizes the degree density $\frac{e(S)}{|S|}$ over all subgraphs $S \subseteq V$. Notice that this is equivalent to maximizing the average degree[1]. The DS-Problem is a powerful primitive for many graph applications including social piggybacking [27], reachability and distance query indexing [19, 37]. However, a large number of applications aims to find subgraphs which are large near-cliques rather than the subgraph that maximizes the average degree. Frequently the DS-Problem fails in finding large near-cliques because it favors large subgraphs with low high edge density f_e. For

[1] In graph theory the term *edge density* refers by default to $f_e = e(S)/\binom{|S|}{2} \in [0, 1]$. However, since direct maximization of f_e is not a meaningful problem (even a single edge achieves the maximum possible edge density), the DS-Problem maximizes the average degree. In the following, we refer to the average degree of a set as its *degree density*.

this reason other formulations have been proposed, see Section 2. Unfortunately, these formulations are **NP**-hard and also inapproximable due the connections with the Maximum Clique problem [32].

The main contribution of this work is a family of tractable formulations which attacks efficiently the problem of extracting large near-cliques and in contrast to well-performing heuristics comes with strong theoretical guarantees. In detail, our contributions are summarized as follows.

New formulation: We introduce the *k-clique densest subgraph problem* (k-Clique-DS-Problem) which generalizes the well-studied DS-Problem [18, 25, 28, 40]. The goal is to maximize the average number of k-cliques induced by a set $S \subseteq V$ over all possible vertex subsets. Notice that the DS-Problem is obtained as a special case for $k = 2$. We introduce also the special case obtained for $k = 3$ as the *triangle densest subgraph problem* (TDS-Problem).

Exact algorithms. We present two exact algorithms for the TDS-Problem. The algorithm which achieves the best running time is based on maximum flow computations and uses $O(n + t)$ space. It is worth outlining that Goldberg's network construction for the DS-Problem which uses $O(n+m)$ space [25, 28] does not generalize to the TDS-Problem. The slower one is based on supermodular maximization and uses linear space $O(n + m)$. Here, n, m, t are the number of vertices, edges and triangles in the input graph. Our algorithms can be modified to yield polynomial time algorithms for the k-Clique-DS-Problem when $k = \Theta(1)$.

Approximation algorithm. We propose a $\frac{1}{3}$-approximation algorithm for the TDS-Problem which runs asymptotically faster than any of the exact algorithms. We also propose a $\frac{1}{3+3\epsilon}$-approximation algorithm for any $\epsilon > 0$ which can be implemented efficiently in MapReduce. The algorithm requires $O(\log(n)/\epsilon)$ rounds and is MapReduce efficient [39] due to the existence of efficient MapReduce triangle counting algorithms, e.g., [48]. Our algorithms can be adapted to the k-Clique-DS-Problem for any $k = \Theta(1)$.

Experimental evaluation. We evaluate our exact and approximation algorithms on numerous real-world networks. Among our findings, we observe that the *optimal triangle densest subgraph* is consistently closer to being a large near-clique compared to the *optimal densest subgraph*. For instance, in the Football network (see Table 1 for a description of the dataset) the DS-Problem returns the whole graph as the densest subgraph, with $f_e = 0.094$ whereas the TDS-Problem returns a subgraph on 18 vertices with $f_e = 0.48$. We also observe that the peeling approximation algorithms for both $k = 2$ and $k = 3$ achieve on real-world networks approximation ratios closer to 1 rather than the pessimistic $\frac{1}{k}$ guarantee.

Graph mining application. We propose a modified version of the TDS-Problem, the constrained triangle densest subgraph problem (Constrained-TDS-Problem), which aims to maximize the triangle density subject to the constraint that the output should contain a specified set of vertices Q. We show how to solve exactly the TDS-Problem. This variation is useful in various data-mining and bioinformatics tasks, see [49].

2. RELATED WORK

Since dense subgraph discovery constitutes a main research topic in graph analysis, a wide variety of related methods exists: heuristics [49, 52, 54], algorithmic contributions on **NP**-hard formulations [5, 12, 22, 40, 49] and poly-time solvable formulations [18, 40, 49]. We focus on the latter.

Densest Subgraph. In the densest subgraph problem we are given a graph G and we wish to find the set $S \subseteq V$ which maximizes the average degree [28, 38]. The densest subgraph can be identified in polynomial time by solving a maximum flow problem [25, 28]. Charikar [18] proved that the greedy algorithm proposed by Asashiro et al. [8] produces a $\frac{1}{2}$-approximation of the densest subgraph in linear time. Asashiro et al. study the complexity of finding dense subgraphs by introducing a generalization of the DS-Problem and the maximum clique problem [7]. A k-core is a maximal connected subgraph of G in which all vertices have degree at least k. It is worth remarking that the same algorithm provides a k-core decomposition of the graph and solves the problem of finding the degeneracy [11]. In the case of directed graphs, the densest subgraph problem is solved in polynomial time as well [18]. Khuller and Saha [40] provide a linear time $\frac{1}{2}$-approximation algorithm for the case of directed graphs among other contributions. Two interesting variations of the *DkS* problem were introduced by Andersen and Chellapilla [5]. The two problems ask for the set S that maximizes the density subject to $|S| \leq k$ (DamkS) and $|S| \geq k$ (DalkS). When restrictions on the size of S are imposed the problem becomes **NP**-hard [40]. Finally, the densest subgraph problem has been considered in various settings, including MapReduce [10], the streaming [10], the dynamic setting [21, 45] and their combination recently [13].

Triangle Counting and Listing. The state of the art algorithm for *exact* triangle counting is due to Alon, Yuster and Zwick [4] and runs in $O(m^{\frac{2\omega}{\omega+1}})$, where currently the fast matrix multiplication exponent ω is 2.3729 [53]. Thus, their algorithm currently runs in $O(m^{1.4081})$ time. The best known listing algorithm until recently was due to Itai and Rodeh [33] which runs in $O(m\alpha(G))$ time, where $\alpha(G)$ is the graph arboricity. Since $\alpha(G) = O(\sqrt{m})$, the running time is always $O(m^{3/2})$. Recently, Björklund, Pagh, Williams and Zwick gave refined algorithms which are output sensitive algorithms [14]. Finally a wealth of approximate triangle counting methods exist [35, 41, 44, 51].

3. PROBLEM DEFINITION

We define the notion of average triangle density.

Definition 1 (Triangle Density). *Let $G(V, E)$ be an undirected graph. For any $S \subseteq V$ we define its triangle density $\tau(S)$ as $\tau(S) = \frac{t(S)}{s}$, where $t(S)$ is the number of triangles induced by S and $s = |S|$.*

Notice that $3\tau(S)$ is the average number of (induced) triangles per vertex in S. The optimization problem we focus on follows.

Problem 1 (TDS-Problem). *Given $G(V, E)$, find a subset of vertices S^* such that $\tau(S^*) = \tau^*_G$ where $\tau^*_G = \max_{S \subseteq V} \tau(S)$.*

It is clear that the DS-Problem and the TDS-Problem in general can result in radically different solutions. Consider for instance a graph G on $2n + 3$ vertices which is the union of a triangle K_3 and of a bipartite clique $K_{n,n}$. The optimal solutions of the DS-Problem and the TDS-Problem are the bipartite clique and the triangle respectively. Therefore, the interesting question is whether maximizing the average degree and the triangle density result in different results in real-world networks. Our results in Section 5 indicate that the answer is positive since the triangle densest subgraph compared to densest subgraph is smaller which exhibits a stronger near-clique structure.

4. PROPOSED METHOD

Section 4.1 provides two algorithms which solve the TDS-Problem exactly. Sections 4.2 and 4.3 provide two approximation algorithms for the TDS-Problem. Finally, Section 4.4 provides a generalization of the DS-Problem and the TDS-Problem to maximizing the average k-clique density and shows how the results from previous Sections adapt to this problem.

4.1 Exact Solutions

Let n, m, t be the number of vertices, edges and triangles in graph G respectively. The algorithm presented in Section 4.1.1 achieves the best running time. We present an algorithm which relies on the supermodularity property of our objective in Section 4.1.2. The latter algorithm, even if slower, requires $O(n+m)$ space, whereas the former $O(n+t)$ space. In real-world networks, typically $m \ll t$. Finally, it is worth mentioning that Charikar's linear program, see §2 in [18], can be extended to a linear program (LP) which solves the TDS-Problem, see [50] for the details.

4.1.1 An $O\big(m^{3/2}+nt+\min{(n,t)}^3\big)$-time exact solution

Algorithm 1 triangle-densest subgraph(G)

```
 1: List the set of triangles $\mathcal{T}(G)$, $t = |\mathcal{T}(G)|$
 2: $l \leftarrow \frac{t}{n}, u \leftarrow \frac{(n-1)(n-2)}{6}$
 3: $S^* \leftarrow \emptyset$
 4: while $u \geq l + \frac{1}{n(n-1)}$ do
 5:     $\alpha \leftarrow \frac{l+u}{2}$
 6:     $H_\alpha \leftarrow$ Construct-Network$(G, \alpha, \mathcal{T}(G))$
 7:     $(S, T) \leftarrow$ minimum $st$-cut in $H_\alpha$
 8:     if $S = \{s\}$ then
 9:         $u \leftarrow \alpha$
10:     else
11:         $l \leftarrow \alpha$
12:         $S^* \leftarrow \big(S \backslash \{s\}\big) \cap V(G)$
13:     end if
14:     Return $S^*$
15: end while
```

Our main theoretical result is the following theorem. Its proof is constructive.

Theorem 1. *There exists an algorithm which which solves the* TDS-Problem *and runs in* $O\big(m^{3/2} + nt + \min{(n,t)}^3\big)$ *time.*

The first term $O(m^{3/2})$ comes from using the Itai-Rodeh algorithm [33] as our triangle listing blackbox. If we use the naive $O(n^3)$ triangle listing algorithm then the running time expression is simplified to $O(n^3 + nt)$. On the other hand, if we use the algorithms of Björklund et al. [14] the first term

Algorithm 2 Construct-Network $(G, \alpha, \mathcal{T}(G))$

```
1: $V(H) \leftarrow \{s\} \cup V(G) \cup \mathcal{T}(G) \cup \{t\}$.
2: For each vertex $v \in V(G)$ add an arc of capacity 1 to
   each triangle $t_i$ it participates in.
3: For each triangle $\Delta = (u, v, w) \in \mathcal{T}(G)$ add arcs to
   $u, v, w$ of capacity 2.
4: Add directed arc $(s, v) \in A(H)$ of capacity $t_v$ for each
   $v \in V(G)$.
5: Add weighted directed arc $(v, t) \in A(H)$ of capacity $3\alpha$
   for each $v \in V(G)$.
6: Return network $H(V(H), A(H), w), s, t \in V(H)$.
```

becomes for dense graphs $\tilde{O}\big(n^{\omega}+n^{3(\omega-1)/(5-\omega)}t^{2(3-\omega)/(5-\omega)}\big)$ and for sparse graphs $\tilde{O}\big(m^{2\omega/(\omega+1)}+m^{3\frac{\omega-1}{\omega+1}}t^{\frac{3-\omega}{\omega+1}}\big)$, where ω is the matrix multiplication exponent. Currently $\omega < 2.3729$ due to [53]. We maintain [33] as our black-box to keep the expressions simpler. However, the reader should keep in mind that the result presented in [14] improves the total running time of the first term.

We work our way to proving Theorem 1 by first proving the next key lemma. Then, we remove the logarithmic factor.

Lemma 1. *Algorithm 1 solves the* TDS-Problem *and runs in* $O\big(m^{3/2} + (nt + \min{(n,t)}^3) \log(n)\big)$ *time.*

Algorithm 1 uses maximum flow computations to solve the TDS-Problem. It is worth outlining that Goldberg's maximum flow algorithm [28] for the DS-Problem is based on a network construction that does not adapt to the case of the TDS-Problem. Algorithm 1 returns an optimal subgraph S^*, i.e., $\tau(S^*) = \tau^*$. The algorithm performs a binary search on the triangle density value α. Specifically, each binary search query corresponds to querying *does there exist a set* $S \subseteq V$ *such that* $t(S)/|S| > \alpha$*?*. For each binary search, we construct a bipartite network H by invoking Algorithm 2. Let $\mathcal{T}(G)$ be the set of triangles in G. The vertex set of H is $V(H) = \{s\} \cup A \cup B \cup \{t\}$, where $A = V(G)$ and $B = \mathcal{T}(G)$. Notice that we overload the notation in order to use the frequently used notation for the sink vertex t. It should always be clear from the context to which entity (number of triangles vs. sink vertex) we refer to. For the purpose of finding $\mathcal{T}(G)$, a triangle listing algorithm is required [14, 33]. The arc set of graph H is created as follows. For each vertex $r \in B$ corresponding to triangle $\Delta(u, v, w)$ we add three incoming and three out-coming arcs. The incoming arcs come from the vertices $u, v, w \in A$ which form triangle $\Delta(u, v, w)$. Each of these arcs has capacity equal to 1. The outgoing arcs go to the same set of vertices u, v, w, but the capacities are equal to 2. In addition to the arcs of capacity 1 from each vertex $u \in A$ to the triangles it participates in, we add an outgoing arc of capacity 3α to the sink vertex t. From the source vertex s we add an outgoing arc to each $u \in A$ of capacity t_v, where t_v is the number of triangles vertex v participates in G. As we have already noticed, H can be constructed in $O(m^{3/2})$ time [33]. It is worth outlining that after computing H for the first time, subsequent networks need to update only the arcs that depend on the parameter α, something not shown in the pseudocode for simplicity. To prove that Algorithm 1 solves the TDS-Problem and runs in $O\big(m^{3/2} + (nt + \min{(n,t)}^3) \log(n)\big)$ time we proceed in steps.

For the sake of the proof, we introduce the following definitions and notation. For a given set of vertices S let $t_i(S)$ be the number of triangles that involve exactly i vertices from S, $i \in \{1,2,3\}$. Notice that $t_3(S)$ is the number of induced triangles by S, for which we have been using the simpler notation $t(S)$ so far.

We use the following claim as our criterion to set the initial values l, u in the binary search.
Claim 1. $\frac{t}{n} \leq \tau(S) \leq \frac{(n-1)(n-2)}{6}$ for any $S \subseteq V$.
The lower bound is obvious since $\tau(V) = \frac{t}{n}$. The upper bound also follows trivially by observing that $\tau(S) \leq \binom{n}{3}/n$ for any $\emptyset \neq S \subseteq V$. This suggests that the optimal value τ^* is always $O(n^2)$.

The next claim serves as a criterion to decide when to stop the binary search.
Claim 2 The smallest possible difference among two distinct values $\tau(S_1), \tau(S_2)$ is equal to $\frac{1}{n(n-1)}$.
To see why, notice that the difference δ between two possible different triangle density values is

$$\delta = \frac{t(S_1)|S_2| - t(S_2)|S_1|}{|S_1||S_2|}.$$

If $|S_1| = |S_2|$ then $|\delta| \geq \frac{1}{n} > \frac{1}{n(n-1)}$, otherwise $|\delta| \geq \frac{1}{|S_1||S_2|} \geq \frac{1}{n(n-1)}$. Notice that combining the above two claims shows that the binary search terminates in at most $\lceil 4 \log n \rceil$ queries. The following lemma is a structural lemma for the optimal $s-t$ cut the network H_α.

Lemma 2. *Consider any minimum st-cut (S,T) in the network H_α. Let $A_1 = S \cap A, B_1 = S \cap B$ and $A_2 = T \cap A, B_2 = T \cap B$. The cost of the min-cut is equal to*

$$\sum_{v \notin A_1} t_v + 2t_2(A_1) + t_1(A_1) + 3\alpha|A_1|.$$

Proof. Case I: $A_1 = \emptyset$: In this case the proposition trivially holds, as the cost is equal to $\sum_{v\in A} t_v = 3t$. It is worth noticing that in this case B_1 has to be also empty, otherwise we contradict the optimality of (S,T). Hence $S = \{s\}, T = A \cup B \cup \{t\}$.

Case II: $A_1 \neq \emptyset$: Consider the cost of the arcs from $A_1 \cup B_1$ to $A_2 \cup B_2$. We consider three different subcases, one per each type of triangle with respect to set A_1.

Type 3: If there exist three vertices $u, v, w \in A_1$ that form a triangle $\Delta(u,v,w)$, then the vertex $r \in B$ corresponding to this specific triangle has to be in B_1. If not, then $r \in B_2$, and we could reduce the cost of the min-cut by 3, if we move the triangle to B_1. Therefore the cost we pay for triangles of type three is 0.

Type 2: Consider three vertices u, v, w such that they form a triangle $\Delta(u,v,w)$ and $u, v \in A_1, w \in A_2$. Then, the vertex $r \in B$ corresponding to this triangle can be either in B_1 or B_2. In both cases we always pay 2 in the cut for each triangle of type two.

Type 1: Finally, in the case u, v, w form a triangle, $u \in A_1, v, w \in A_2$ the vertex $r \in B$ corresponding to triangle $\Delta(u,v,w)$ will be in B_2. If not, then it lies in B_1 and we could decrease the cost of the cut by 3 if we move it in B_2. Hence, we pay 1 in the cut for each triangle of type one.

Therefore the cost due to the various types of triangles with respect to A_1 is equal to $2t_2(A_1) + t_1(A_1)$.

Furthermore, the cost of the arcs from source s to T is equal to $\sum_{v\in A_2} t_v = \sum_{v\notin A_1} t_v$. The cost of the arcs from A_1 to T is equal to $3\alpha|A_1|$. Summing up the individual cost terms, we obtain that the total cost is equal to $\sum_{v\notin A_1} t_v + 2t_2(A_1) + t_1(A_1) + 3\alpha|A_1|$. □

The next lemma proves the correctness of the binary search in Algorithm 1.

Lemma 3. *(a) If there exists a set $W \subseteq V(G)$ such that $t_3(W) > \alpha|W|$ then any minimum st-cut (S,T) in H_α satisfies $S \backslash \{s\} \neq \emptyset$. (b) Furthermore, if there does not exists a set W such that $t_3(W) > \alpha|W|$ then the cut $(\{s\}, A \cup B \cup \{t\})$ is a minimum st-cut.*

Proof. (a) Let $W \subseteq V$ be such that

$$t_3(W) > \alpha|W|. \tag{1}$$

Suppose for the sake of contradiction that the minimum st-cut is achieved by $(\{s\}, A \cup B \cup \{t\})$. In this case the cost of the minimum st-cut is $\sum_{v\in A} t_v = 3t$. Now, consider the following (S,T) cut. Set S consists of the source vertex s, $A_1 = W$ and B_1 be the set of triangles of type 3 and 2 induced by A_1. Let T be the rest of the vertices in H. The cost of this cut is

$$\mathit{cap}(S,T) = \sum_{v\notin A_1} t_v + 2t_2(A_1) + t_1(A_1) + 3\alpha|A_1|.$$

Therefore, by our assumption that the minimum st-cut is achieved by $(\{s\}, A \cup B \cup \{t\})$ we obtain

$$3t \leq \sum_{v\notin A_1} t_v + 2t_2(A_1) + t_1(A_1) + 3\alpha|A_1|. \tag{2}$$

Now, notice that by double counting

$$\sum_{v\in A_1} t_v = 3t_3(A_1) + 2t_2(A_1) + t_1(A_1).$$

Furthermore, we observe

$$\sum_{v\in A_1} t_v + \sum_{v\notin A_1} t_v = 3t.$$

By combining these two facts, and the fact that $3t$ is the capacity of the minimum cut, we obtain the following contradiction of Inequality (1).

$$3t \leq \sum_{v\notin A_1} t_v + 2t_2(A_1) + t_1(A_1) + 3\alpha|A_1| \Leftrightarrow t_3(W) \leq \alpha|W|.$$

(b) By Lemma 2, for any minimum st-cut (S,T) the capacity of the cut is equal to $\sum_{v\notin A_1} t_v + 2t_2(A_1) + t_1(A_1) + 3\alpha|A_1|$, where $A_1 = A \cap S, A_2 = A \cap T$. Suppose for the sake of contradiction that the cut $(\{s\}, A \cup B \cup \{t\})$ is not a minimum cut. Therefore,

$$\mathit{cap}(\{s\}, A \cup B \cup \{t\}) = 3t > \sum_{v\notin A_1} t_v + 2t_2(A_1) + t_1(A_1) + 3\alpha|A_1|.$$

Using the same algebraic analysis as in (a), the above statement implies the contradiction $t_3(W) > \alpha|W|$, where $W = A_1$. □

Now we can complete the proof of Lemma 1.

Proof. The termination of Algorithm 1 follows directly from Claims 1, 2. The correctness follows from Lemmata 2, 3. The running time follows from Claims 1,2 which show that the number of binary search queries is $O(\log(n))$ and each binary search query can be performed in $O\big(nt + \min(n,t)^3\big)$ time using the algorithm due to Ahuja, Orlin, Stein and Tarjan [3][2] or Gusfield's algorithm [31]. □

The proof of Theorem 1 follows from Lemma 1 and the fact that the parametric maximum flow algorithm of Ahuja, Orlin, Stein and Tarjan [3], see also [25], s(h)aves the logarithmic factor from the running time.

4.1.2 An $O\big((n^5m^{1.4081} + n^6)\big)\log(n))$-time exact solution

In this Section we provide a second exact algorithm for the TDS-Problem. First, we provide the necessary theoretical background.

Definition 2 (Supermodular function). *Let V be a finite set. The set function $f : 2^V \to \mathbb{R}$ is supermodular if and only if for all $A, B \subseteq V$*

$$f(A \cup B) \geq f(A) + f(B) - f(A \cap B).$$

A function f is supermodular if and only if $-f$ is submodular.

Sub- and supermodular functions constitute an important class of functions with various special properties. In this work, we are primarily interested in the fact that *maximizing a supermodular function is solvable in strongly polynomial time* [30, 34, 42, 46]. For our purposes, we state the following result which we use as a subroutine in our proposed algorithm.

Theorem 2 ([43]). *There exists an algorithm for maximizing an integer valued supermodular function f which runs in $O\big(n^5 EO + n^6\big)$ time, where $n = |V|$ is the size of the ground set V and EO is the maximum amount of time to evaluate $f(S)$ for a subset $S \subseteq V$.*

We show in the following that when the ground set is the set of vertices V and $f_\alpha : 2^V \to \mathbb{R}$ is defined by $f_\alpha(S) = t(S) - \alpha|S|$ where $\alpha \in \mathbb{R}^+$, we can solve the TDS-Problem in polynomial time.

Theorem 3. *Function $f : V \to \mathbb{R}$ where $f(S) = t(S) - \alpha|S|$ is supermodular.*

Proof. Let $A, B \subseteq V$. Let $t : 2^V \to \mathbb{R}$ be the function which for each set of vertices S returns the number of induced triangles $t(S)$. By careful counting

$$t(A \cup B) = t(A) + t(B) - t(A \cap B) + t_1(A : B\backslash A) + t_2(A : B\backslash A),$$

where $t_1(A : B\backslash A), t_2(A : B\backslash A)$ are the number of triangle with one, two vertices in A and two, one vertices in $B\backslash A$ respectively. Hence, for any $A, B \subseteq V$

$$t(A \cup B) + t(A \cap B) \geq t(A) + t(B)$$

and the function t is supermodular. Furthermore, for any $\alpha > 0$ the function $-\alpha|S|$ is supermodular. Since the sum of two supermodular functions is supermodular, the result follows. □

Theorem 3 naturally suggests Algorithm 3. The algorithm will run in a logarithmic number of rounds. In each round we maximize function f_α using Orlin-Supermodular-Opt which takes as input arguments the graph G and the parameter $\alpha > 0$ [43]. We assume for simplicity that within the procedure Orlin-Supermodular-Opt function f is evaluated using an efficient exact triangle counting algorithm [4]. The algorithm of Alon, Yuster and Zwick [4] runs in $O(m^{2\omega/(\omega+1)})$ time where $\omega < 2.3729$ [53]. This suggests the $EO = O(m^{1.4081})$. The overall running time of Algorithm 3 is $O\big((n^5m^{1.4081} + n^6)\log(n)\big)$ and the space usage $O(n+m)$ rather than $O(n+t)$.

Algorithm 3 triangle-densest subgraph(G) [Supermodularity]

```
1: $l \leftarrow 0, u \leftarrow \frac{(n-1)(n-2)}{6}, S^* \leftarrow V$
2: while $u \geq l + \frac{1}{n(n-1)}$ do
3:     $\alpha \leftarrow \frac{l+u}{2}$
4:     $(val, S) \leftarrow$ Orlin-Supermodular-Opt$(G, \alpha)$
5:     if $val < 0$ then
6:         $u \leftarrow \alpha$
7:     else
8:         $l \leftarrow \alpha$
9:         $S^* \leftarrow S$
10:    end if
11:    Return $S^*$
12: end while
```

4.2 A $\frac{1}{3}$-approximation algorithm

In this Section we provide an algorithm for the TDS-Problem which provides a $\frac{1}{3}$-approximation. Our algorithm follows the peeling paradigm, see [8, 18, 40, 36]. Specifically, in each round it removes the vertex which participates in the smallest number of triangles and returns the subgraph that achieves the largest triangle density. The pseudocode is shown in Algorithm 4.

Algorithm 4 Peel-Triangles(G)

```
1: Count the number of triangles $t_v$ for each vertex $v \in V$
2: $H_n \leftarrow G$
3: for $i \leftarrow n$ to 2 do
4:     Let $v$ be the vertex of $G_i$ of minimum number of triangles
5:     $H_{i-1} \leftarrow H_i \backslash v$
6: end for
7: Return $H_j$ that achieves maximum triangle density among $H_i$s, $i = 1, \ldots, n$.
```

Theorem 4. *Algorithm 4 is a $\frac{1}{3}$-approximation algorithm for the* TDS-Problem.

[2]Notice that the network H_α has $O(n+t)$ arcs, therefore the running time of [3] is $O(\min(n,t)(n+t) + \min(n,t)^3) = O(nt + \min(n,t)^3)$.

Proof. Let S^* be an optimal set. Let $v \in S^*, |S^*| = s^*$ and $t_A(v)$ be the number of induced triangles by A that v participates in. Then,

$$\tau_G^* = \frac{t(S^*)}{s^*} \geq \frac{t(S^* \backslash \{v\})}{s^*-1} \Leftrightarrow t_{S^*}(v) \geq \tau_G^*,$$

since $t(S^* \backslash \{v\}) = t(S^*) - t_{S^*}(v)$. Consider the iteration before the algorithm removes the first vertex v that belongs in S^*. Call the set of vertices W. Clearly, $S^* \subseteq W$ and for each vertex $u \in W$ the following lower bound holds $t_W(u) \geq t_W(v) \geq t_{S^*}(v) \geq \tau_G^*$ due to the greediness of Algorithm 3. This provides a lower bound on the total number of triangles induced by W

$$t(W) = \frac{1}{3}\sum_{u \in W} t_W(u) \geq \frac{1}{3}|W|\tau_G^* \Rightarrow \frac{t(W)}{|W|} \geq \frac{1}{3}\tau_G^*.$$

To complete the proof, notice that the algorithm returns a subgraph S such that $\tau(S) \geq \tau(W) \geq \frac{1}{3}\tau_G^*$. □

The key difference compared to the DS-Problem peeling algorithm [18] is that when we remove a vertex, the counts of its neighbors may decrease more than 1. Therefore, when vertex v is removed, we update the counts of its neighbors in $O\Big(\binom{deg(v)}{2}\Big)$ time, by looking how many triangles each of its neighbors has after v is removed. Notice that $O\big(\sum_v \binom{deg(v)}{2}\big) = O(mn)$.

4.3 MapReduce Implementation

The MapReduce framework [20] has become the *de facto* standard for processing large-scale datasets. In the following, we show how we can approximate efficiently the TDS-Problem in MapReduce. Before we describe the algorithm, we show that Algorithm 5 for any $\epsilon > 0$ terminates and provides a $\frac{1}{3+3\epsilon}$-approximation. The idea behind this algorithm is to peel vertices in batches [10, 29] rather than one by one.

Algorithm 5 `Peel-Triangles-in-Batches`$(G, \epsilon > 0)$

```
1: $S_{out}, S \leftarrow V$
2: while  $S \neq \emptyset$ do
3:     $A(S) \leftarrow \{i \in S : t_S(i) \leq 3(1+\epsilon)\tau(S)\}$
4:     $S \leftarrow S \backslash A(S)$
5:     if $\tau(S) \geq \tau(S_{out})$ then
6:         $S_{out} \leftarrow S$
7:     end if
8: end while
9: Return $S_{out}$.
```

Lemma 4. *For any $\epsilon > 0$, Algorithm 5 provides a $\frac{1}{(3+3\epsilon)}$-approximation to the* TDS-Problem. *Furthermore, it terminates in $O(\log_{1+\epsilon}(n))$ passes.*

Proof. Let S^* be an optimal solution to the TDS-Problem. As we proved in Theorem 4, for any $v \in S^*$ it is true that $t_{S^*}(v) \geq \tau_G^*$. Furthermore, in each round at least one vertex is removed. To see why, assume for the sake of contradiction that $A(S) = \emptyset$ for some S during the execution of the algorithm. Then, we obtain the contradiction that $3|S|\tau(S) = \sum_{v \in S} t_S(v) \geq (3+3\epsilon)|S|\tau(S)$. Consider the round where the algorithm for the first time removes a vertex $v \in S^*$. Let W be the corresponding set of vertices. Since $v \in A(W)$ is peeled off, we obtain an upper bound on its induced degree, namely $v \in A(W) \Rightarrow t_W(v) \leq (3+3\epsilon)\tau(W)$. Since $S^* \subseteq W$, we obtain

$$(3+3\epsilon)\tau(W) \geq t_W(v) \geq t_{S^*}(v) \geq \tau(S^*),$$

which proves that Algorithm 5 is a $\frac{1}{(3+3\epsilon)}$-approximation to the TDS-Problem. To see why the algorithm terminates in logarithmic number of rounds, notice that

$$3t(S) > \sum_{v \in S \backslash A(S)} t_S(v) \geq (3+3\epsilon)\big(|S| - |A(S)|\big)\frac{t(S)}{|S|} \Leftrightarrow$$

$$|A(S)| \geq \frac{\epsilon}{1+\epsilon}|S| \Leftrightarrow |S \backslash A(S)| \leq \frac{1}{1+\epsilon}|S|.$$

Since S decreases by a factor of $\frac{1}{1+\epsilon}$ in each round, the algorithm terminates in $O(\log_{1+\epsilon}(n)) = O\big(\frac{\log(n)}{\epsilon}\big)$ rounds. □

MapReduce *Implementation:* Now we are able to describe our algorithm in MapReduce. It uses any of the efficient algorithms of Suri and Vassilvitski [48] as a subroutine to count triangles per vertex in each round. The removal of the vertices which participate in less triangles than the threshold, is done in two rounds, as in [10]. For completeness, we describe the procedure here. The set of vertices S to be peeled off in each round are marked by adding a key-value pair $\langle v; \$\rangle$ for each $v \in S$. Each edge (u, v) is mapped to $\langle u; v\rangle$. The reducer receives all endpoints of the edges incident with v and the symbol \$ in case the vertex is marked for deletion. In case the vertex is marked, then the reduce task returns nothing, otherwise it copies its input. In the second round, we perform the same procedure with the only difference being that we map each edge (u, v) to $\langle v; u\rangle$. Therefore, the edges which remain have both endpoints unmarked. The algorithm runs in $O(\log(n)/\epsilon)$, as it takes $O(\log(n)/\epsilon)$ peeling off rounds, and in each peeling round, constant number of rounds is needed to count triangles per vertex, mark vertices for deletion and remove the corresponding vertex set.

4.4 k-clique Densest Subgraph

We outline that our proposed methods can be adapted to the following generalization of the DS-Problem and the TDS-Problem.

Definition 3 (k-clique-densest subgraph). *Let $G(V, E)$ be an undirected graph. For any $S \subseteq V$ we define its k-clique density $h_k(S)$, $k \geq 2$ as $h_k(S) = \frac{c_k(S)}{s}$, where $c_k(S)$ is the number of k-cliques induced by S and $s = |S|$.*

Problem 2 (k-Clique-DS-Problem). *Given $G(V, E)$, find a subset of vertices S^* such that $h_k(S^*) = h_k^*$ where $h_k^* = \max_{S \subseteq V} h_k(S)$.*

As in the triangle densest subgraph problem, we create a network H parameterized by the value α on which we perform our binary search. The procedure is described in Algorithm 6. The set $\mathcal{C}(G)$ is the set of k-cliques in G. We then invoke Algorithm 1, with the upper bound u set to n^k. Following the analysis of Theorem 1, we see that the k-Clique-DS-Problem is solvable in polynomial time. For instance, using Gusfield's algorithm [31] or [3] in each binary search query we get an overall running time $O\big(n^k + (n|\mathcal{C}(G)| + n^3)\log(n)\big) = O(n^{k+1}\log(n))$. Using the improved result due to Ahuja, Orlin, Stein and Tarjan for parametric max flows in unbalanced bipartite graphs [3], we save the logarithmic factor in the running time.

Algorithm 6 `Construct-Network-k` $(G, \alpha, \mathcal{C}(G), k)$

```
1: $V(H) \leftarrow \{s\} \cup V(G) \cup \mathcal{C}(G) \cup \{t\}$.
2: For each vertex $v \in V(G)$ add an arc of capacity 1 to
   each $k$-clique $c_i$ it participates in.
3: For each $k$-clique $(u_{i_1}, \ldots, u_{i_k}) \in \mathcal{C}(G)$ add arcs to
   $u_{i_1}, \ldots, u_{i_k}$ of capacity $k-1$.
4: Add directed arc $(s, v) \in A(H)$ of capacity $c_v$ for each
   $v \in V(G)$.
5: Add weighted directed arc $(v, t) \in A(H)$ of capacity $k\alpha$
   for each $v \in V(G)$.
6: Return network $H(V(H), A(H), w), s, t \in V(H)$.
```

Furthermore, Algorithm 4 can also be modified, by removing in each round the vertex with the smallest number of k-cliques, to obtain Corollary 2. As the analogy of Theorem 4.

Corollary 1. *The algorithm which peels off in each round the vertex with the minimum number of k-cliques and returns the subgraph that achieves the largest k-clique density, is a $\frac{1}{k}$-approximation algorithm for the* k-Clique-DS-Problem.

Similarly, Algorithm 5 and the MapReduce implementation can be modified to solve the k-Clique-DS-Problem. We omit the details.

Corollary 2. *The algorithm which peels off in each round the set of vertices with less than $k(1+\epsilon)h(S)$, where $h(S)$ is the k-clique density in that round, terminates in $O(\log_{1+\epsilon}(n))$ rounds and provides a $\frac{1}{k(1+\epsilon)}$-approximation guarantee for the* k-Clique-DS-Problem. *Furthermore, using [23], we obtain an efficient* MapReduce *implementation.*

We illustrate an example where choosing a larger k value yields benefits. Let $G \sim G(n,p)$ be an Erdös-Rényi graph, where $p = p(n)$. Assume that we plant a clique K of size n^γ for some constant $\gamma > 0$. We wish to show a non-trivial range of $p = p(n)$ values such that the following conditions hold: $h_2(C) = \frac{|E(K)|}{|K|} = \frac{\binom{n^\gamma}{2}}{n^\gamma} < \frac{p\binom{n}{2}}{n} = \mathbb{E}\left[h_2(V)\right]$, and for $k \geq 3$ $h_k(C) = \frac{\binom{n^\gamma}{k}}{n^\gamma} > \frac{p^{\binom{k}{2}}\binom{n}{k}}{n} = \mathbb{E}\left[h_k(V)\right]$.
By simple algebraic manipulation we see that p satisfies both conditions if $O\big(n^{-(1-\gamma)}\big) < p < O\big(n^{-\frac{2}{k}(1-\gamma)}\big)$[3]. Clearly, for larger k values, we allow ourselves a larger range of p values for which we can find the hidden clique in expectation.

Our preliminary experimental results for $k = 4$ indicate that the 4-clique-densest subgraph gets closer to a large near-clique compared to the triangle densest subgraph. However, the gain of moving from the densest subgraph to the triangle densest subgraph with respect to extracting large near-cliques is larger than the gain of moving from the triangle densest subgraph to the 4-clique-densest subgraph.

[3] Notice that for this range of p, the graph is connected and the clique number is constant with high probability [15]

5. EXPERIMENTAL EVALUATION

Before we present our findings in detail, we summarize them: (i) the TDS-Problem and the proposed algorithms constitute new valuable graph mining primitives for finding large near-cliques, (ii) the $\frac{1}{3}$-approximation algorithm (Algorithm 4) achieves significantly better approximations than the pessimistic $\frac{1}{3}$ guarantee. Also it is significantly faster. (iii) Trying a small range of ϵ values for Algorithm 5 is in general a safer strategy compared to running experiments with fixed choice.

5.1 Experimental Setup

The datasets we use are shown in Table 1. All graphs were made simple and undirected by ignoring the edge direction, when the graph is directed, and removing self-loops and multiple edges, if any. The experiments were performed on a single machine, with Intel(R) Core(TM) i5 CPU at 2.40 GHz, with 3.86GB of main memory. We have implemented Algorithm 1 in Matlab R2011a using a maximum flow implementation due to Kolmogorov and Boykov [16] as our subroutine which runs in time $O(t(n+t)^3)$. This implementation is prohibitively expensive even for small graphs which have a large number of triangles. We have coded the peeling algorithm in C++ using priority queues. As our triangle listing algorithm, we use the simple node iterator algorithm which checks for each vertex the number of edges among its neighbors. The code is publicly available at `http://people.seas.harvard.edu/~babis/code.html`. We measure the quality of each extracted subgraph by two measures: the edge density of the extracted subgraph $f_e = e(S)/\binom{|S|}{2}$ and the output size $|S|$. Notice that when f_e is close to 1, the extracted subgraph is close to being a clique.

5.2 Main Findings

Table 2 shows the results obtained on several popular small- and medium-sized graphs. Each column corresponds to a dataset. The rows correspond to measurements for each method we use to extract a subgraph. Specifically, the first (DS), second ($\frac{1}{2}$-DS), third (TDS) and fourth ($\frac{1}{3}$-TDS) rows correspond to the subgraph extracted by Goldberg's exact algorithm [28] for the DS-Problem, Charikar's $\frac{1}{2}$-approximation algorithm [18] for the DS-Problem, Algorithm 1 and Algorithm 4 for the TDS-Problem respectively. For each optimal extracted subgraph S, we show its size as a fraction of the total number of vertices, the edge density $f_e(S)$, the average degree $2\delta(S) = 2e(S)/|S|$ and the average number of triangles per vertex $3\tau(S) = 3t(S)/|S|$.

We observe that the triangle-densest subgraph is closer to being a near-clique compared to the densest subgraph. A pronounced example is the Football network where the optimal densest subgraph is the whole network with $f_e = 0.0094$, whereas the optimal triangle-densest subgraph is a set of 18 vertices with edge density 0.48. Finally, we observe that the quality of Algorithm's 4 output is very close to the optimal solution and sometimes even better. The same observation holds for the case of Charikar's $\frac{1}{2}$-approximation algorithm [18].

We use the C++ implementation of Algorithm 4 and a C++ implementation of Charikar's $\frac{1}{2}$-approximation algorithm on the rest of the datasets of Table 1. The results are shown in Table 3, up to two decimal digits of accuracy. The ca-HepTh dataset is an exception, as the optimal solutions coincide. We also notice that the run times appear the same

Name	Nodes	Edges	Description
Adjnoun	112	425	Generated by processing text data
AS-735	6 475	12 572	Autonomous Systems
AS-caida	26 475	53 381	Autonomous Systems
ca-Astro	17 903	196 972	Co-authorship
ca-GrQC	4 158	13 422	Co-authorship
ca-HepTh	11 204	117 619	Co-authorship
Celegans	297	4 296	Neural network of C. Elegans
DBLP	53 442	255 936	Co-authorship
Epinions	75 877	405 739	Social network
Enron	33 696	180 811	Email
EuAll	224 832	339 925	Email
Football	115	613	NCAA football game network
Karate	34	78	Social network
Lesmis	77	254	Generated by processing text data
Political blogs	1 490	16 715	Generated by processing sales data
Political books	105	441	Blog network
soc-Slashdot0811	77 360	469 180	Person to Person
soc-Slashdot0902	82 168	504 230	Person to Person
wb-cs-Stanford	8 929	26 320	Web Graph
web-Google	855 802	4 291 352	Web Graph
web-NotreDame	325 729	1 090 108	Web Graph
Wiki-vote	7 066	100 736	Wikipedia "who-votes-whom"

Table 1: Datasets used in our experiments.

Method	Measure	Adjnoun	Celegans	Football	Karate	Lesmis	Polblogs	Polbooks
DS	$\frac{\|S\|}{\|V\|}(\%)$	42.86	45.8	100	47.1	29.9	19.1	51.4
	2δ	9.58	17.16	10.66	5.25	10.78	55.82	9.40
	f_e	0.20	0.13	0.094	0.35	0.49	0.196	0.18
	3τ	14	45.93	21.12	5.64	41.61	768.87	22.68
$\frac{1}{2}$-DS	$\frac{\|S\|}{\|V\|}(\%)$	41.1	42.4	100	52.9	29.9	18.7	57.1
	2δ	9.57	17.1	10.66	5.2	10.78	55.8	9.3
	f_e	0.21	0.14	0.094	0.31	0.49	0.20	0.16
	3τ	14.16	46.5	21.12	5.16	41.61	774.6	22.68
TDS	$\frac{\|S\|}{\|V\|}(\%)$	36.6	10.4	15.7	17.7	16.9	8.1	19.1
	2δ	9.37	13.81	8.22	4.67	10.62	55.72	9.34
	f_e	0.23	0.46	0.48	0.93	0.89	0.46	0.50
	3τ	15	56.82	28	8.01	47.31	972.36	25.95
$\frac{1}{3}$-TDS	$\frac{\|S\|}{\|V\|}(\%)$	36.6	9.1	15.7	17.7	16.9	8.1	15.2
	2δ	9.37	13.56	8.22	4.67	10.62	55.72	9.13
	f_e	0.23	0.52	0.48	0.93	0.89	0.46	0.61
	3τ	15	56.55	28	8.01	47.31	972.36	25.5

Table 2: Comparison of the extracted subgraphs by the Goldberg's exact algorithm for the DS-Problem (DS), Charikar's $\frac{1}{2}$-approximation algorithm ($\frac{1}{2}$-DS), our exact algorithm for the TDS-Problem (TDS) and our $\frac{1}{3}$-approximation algorithm ($\frac{1}{2}$-TDS). Here, $f_e(S) = e(S)/\binom{|S|}{2}$ is the edge density of the extracted subgraph, $2\delta(S) = 2e(S)/|S|$ is the average degree and $3\tau(S) = 3t(S)/|S|$ is the average number of triangles.

for ca-HepTh due to using two decimal digits of accuracy. On other datasets, we observe differences between the two solutions. For instance, for the collaboration network ca-Astro the densest subgraph is a subgraph with 1 184 vertices and $f_e = 0.05$. The triangle-densest subgraph is a clique with 57 vertices. Overall, we verify the fact that the triangle-densest subgraph is closer to being a near-clique. Finally, the run times are shown. Notice that the run times reported for for Algorithm 4 include both the triangle counting and the peeling phase.

5.3 Exploring parameter ϵ in Algorithm 5

In this Section we present the results of Algorithm 5 on the DBLP graph. This is particularly interesting instance as it indicates that instead of trying to select a good ϵ value, it is worth trying out at least few values, assuming computational resources are available. We range ϵ from 0.1 to 1.8 with a step of 0.1. Figure 1(a) plots the number of rounds Algorithm 5 takes to terminate as a function of ϵ. We observe that even for small ϵ values the number of rounds is 6. The reader should compare this to the upper bound predicted by Lemma 4 which exceeds 100. Figure 1(b) plots the

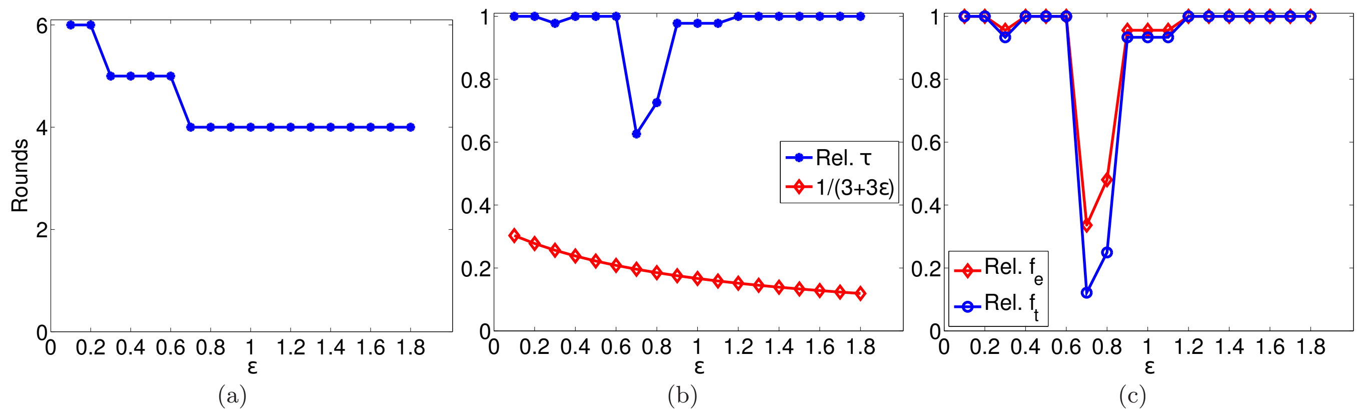

Figure 1: Exploring the trade-off between the number of rounds and accuracy as a function of the parameter ϵ for Algorithm 5. Let S, S^* be the extracted subgraphs by Algorithms 5 and 1 respectively. (a) Number of rounds, (b) relative average triangle density ratio $\frac{\tau(S)}{\tau^*}$ (blue $*$) and the approximation guarantee $1/(3+3\epsilon)$ (red $\diamond$), and (c) relative ratios $\frac{f_e(S)}{f_e(S^*)}, \frac{f_t(S)}{f_t(S^*)}$ as functions of ϵ.

	$\frac{1}{2}$-DS			$\frac{1}{3}$-TDS		
	$\|S\|$	f_e	T	$\|S\|$	f_e	T
AS-735	59	0.28	0.00	13	0.8	0.07
AS-caida	143	0.14	0.02	27	0.52	0.63
ca-Astro	1 184	0.05	0.06	57	1	1.42
ca-GrQC	42	0.79	0.00	14	0.89	0.02
ca-HepTh	32	1	0.02	32	1	0.02
Epinions	999	0.121	0.08	431	0.256	8.75
Enron	555	0.14	0.02	390	0.19	2.01
EuAll	507	0.13	0.08	200	0.29	9.52
soc-Slashdot0811	207	0.41	0.13	253	0.49	6.85
soc-Slashdot0902	219	0.40	0.16	173	0.50	7.72
wb-cs-Stanford	84	0.64	0.48	26	0.80	0.67
web-Google	240	0.23	2.54	120	0.44	79.5
web-NotreDame	1 367	0.115	0.50	457	0.345	16.3
Wiki-vote	846	0.11	0.00	464	0.80	0.19

Table 3: Comparison of the extracted subgraphs by the $\frac{1}{2}$-approximation algorithm of Charikar and the $\frac{1}{3}$-approximation algorithm, Algorithm 4. The respective run times are shown in seconds.

relative ratio $Rel.\ \tau = \frac{\tau(S)}{\tau^*}$ where S is the output of Algorithm 5. For convenience, the lower bound $\frac{1}{3+3\epsilon}$ is plotted with red color.

Besides the ratio $\frac{f_e(S)}{f_e(S^*)}$, figure 1(c) plots also the relative ratio $\frac{f_t(S)}{f_t(S^*)}$ as a function of ϵ. Here $f_t(S) = \frac{t(S)}{\binom{|S|}{3}}$. As we observe, the quality of Algorithm 5 is close to the optimal solution except for $\epsilon = 0.7$ and $\epsilon = 0.8$. By inspecting why this happens we observe that the optimal triangle-densest subgraph is a clique of 44 vertices. It turns out that for $\epsilon = 0.7, 0.8$ the optimal subgraph which is found in the last round of the execution of the algorithm (the latter happens for all ϵ values) consists of 98 and 74 vertices which contain as a subgraph the optimal K_{44}. For other values of ϵ, the subgraph in the last round is either the optimal K_{44} or close to it, with few more extra vertices. This example shows the potential danger of using a single value for ϵ, suggesting that trying out a small number of ϵ values can be significantly beneficial in terms of the approximation quality.

6. APPLICATION: ORGANIZING COCKTAIL PARTIES

A graph mining problem that comes up in various applications is the following: given a set of vertices $Q \subseteq V$, find a dense subgraph containing Q. We refer to this type of graph mining problems as *cocktail problems*, due to the following motivation, c.f. [47]. Suppose that a set of people Q wants to organize a cocktail party. How do they invite other people to the party so that the set of all the participants, including Q, are as similar as possible? A variation of the TDS-Problem which addresses this graph mining problem follows.

Problem 3 (Constrained-TDS-Problem). *Given a graph $G(V, E)$ and $Q \subseteq V$, find the subset of vertices S^* that maximizes the triangle density such that $Q \subseteq S^*$,*

$$S^* = \arg \max_{Q \subseteq S \subseteq V} \tau(S).$$

The Constrained-TDS-Problem can be solved by modifying our proposed algorithms accordingly. A useful corollary follows.

Corollary 3. *The* Constrained-TDS-Problem *is solvable in polynomial time by adding arcs from s to $v \in A$ of large enough capacities, e.g., capacities equal to $n^3 + 1$ are sufficiently large. Furthermore, the peeling algorithm which avoids removing vertices from Q is a $\frac{1}{3}$-approximation algorithm for the* Constrained-TDS-Problem.

In the following we evaluate the $\frac{1}{3}$-approximation algorithm on two datasets. The two experiments indicate two different types of performances that should be expected in real-world applications. The first is a positive whereas the second is negative case. Both experiments here serve as sanity checks[4]

Political vote data. We obtain Senate data for the first session (2006) of the 109th congress which spanned the period from January 3, 2005 to January 3, 2007, during the

[4] For instance, by preprocessing the political vote data from a matrix form to a graph using a threshold for edge additions, results in information loss.

fifth and sixth years of George W. Bush's presidency [1]. In this Congress, there were 55, 45 and 1 Republican, Democratic and independent senators respectively. The dataset can be downloaded from the US Senate web page http://www.senate.gov. We preprocess the dataset in the following way: we add an edge between two senators if among the bills for which they both casted a vote, they voted at least 80% of the times in the same way. The resulting graph has 100 vertices and 2034 edges. We run the $\frac{1}{3}$-approximation algorithm on this graph using as our set Q the first three republicans according to lexicographic order: Alexander (R-TN), Allard (R-CO) and Allen (R-VA). We obtain at our output a subgraph consisting of 47 vertices. By inspecting their party, we find that 100% of them are Republicans. This shows that our algorithm in this case succeeds in finding the large majority of the cluster of republicans. It is interesting that the 8 remaining Republicans do not enter the triangle-densest subgraph. A careful inspection of the data, c.f. [2], indicates that 6 republicans agree with the party vote on at most 79% of the bills, and 8 of them on at most 85% of the bills.

DBLP graph. We input as a query set Q a set of scientists who have established themselves in theory and algorithm design: Richard Karp, Christos Papadimitriou, Mihalis Yannakakis and Santosh Vempala. The algorithm returns at its output the query set and a set S of 44 vertices corresponding to a clique of (mostly) Italian computer scientists. We list a subset of the 44 vertices here: M. Bencivenni, M. Canaparo, F. Capannini, L. Carota, M. Carpene, R. Veraldi, P. Veronesi, M. Vistoli, R. Zappi. The output graph induced by $S \cup Q$ is disconnected. Therefore, this can be easily explained because of the following (folklore) inequality, given that $|Q| < |S|$ in our example.

Claim 1. *Let a, b, c, d be non-negative. Then,*

$$\max\left(\frac{a}{c}, \frac{b}{d}\right) \geq \frac{a+b}{c+d} \geq \min\left(\frac{a}{c}, \frac{b}{d}\right) \qquad (3)$$

In our example, we get $a = t(S), c = |S|, b = t(Q), d = |Q|$. In such a scenario, where the output consists of the union of a dense subgraph and the query set Q, an algorithm which builds itself up from Q -assuming Q is not an independent set- to V by adding vertices which create as many triangles as possible and returning the maximum density subgraph, rather than peeling vertices from V down to Q should be preferred in practice, see also [49].

7. CONCLUSION

In this work we introduce the average triangle density as a novel objective for attacking the important problem of finding near-cliques. We propose exact and approximation algorithms. Furthermore, our techniques can solve the more general problem of maximizing the k-clique density. Experimentally we verify the value of the TDS-Problem as a novel addition to the graph mining toolbox.

Our work leaves numerous problems open, including the following: (a) Can we obtain a faster exact algorithm by improving the space usage of the network construction? (b) Can we use sparsification to obtain faster approximate solutions [44]?

8. REFERENCES

[1] http://tinyurl.com/bwgpka. 6

[2] http://tinyurl.com/zgdam. 6

[3] R. K. Ahuja, J. B. Orlin, C. Stein, and R. E. Tarjan. Improved algorithms for bipartite network flow. *SIAM Journal on Computing*, 23(5):906–933, 1994. 4.1.1, 2, 4.4

[4] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. 2, 4.1.2

[5] R. Andersen and K. Chellapilla. Finding dense subgraphs with size bounds. In *WAW*, 2009. 2

[6] A. Angel, N. Sarkas, N. Koudas, and D. Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *Proc. VLDB Endow.*, 5(6):574–585, Feb. 2012. 1

[7] Y. Asahiro, R. Hassin, and K. Iwama. Complexity of finding dense subgraphs. *Discr. Ap. Math.*, 121(1-3), 2002. 2

[8] Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama. Greedily finding a dense subgraph. *J. Algorithms*, 34(2), 2000. (document), 2, 4.2

[9] G. D. Bader and C. W. Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC bioinformatics*, 4(1):2, 2003. 1

[10] B. Bahmani, R. Kumar, and S. Vassilvitskii. Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment*, 5(5):454–465, 2012. (document), 2, 4.3, 4.3

[11] V. Batagelj and M. Zaversnik. An O(m) algorithm for cores decomposition of networks. *Arxiv*, arXiv.cs/0310049, 2003. 2

[12] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan. Detecting high log-densities: an $O(n^{1/4})$ approximation for densest k-subgraph. In *Proceedings of the 42nd ACM symposium on Theory of computing*, pages 201–210. ACM, 2010. 2

[13] S. Bhattacharya, M. Henziger, D. Nanongkai, and C.E. Tsourakakis. Space- and time-efficient algorithms for maintaining dense subgraphs on one-pass dynamic streams. In *Proceedings of the 47th ACM symposium on Theory of computing*, 2015 2

[14] A. Björklund, R. Pagh, V. Williams Vassilevska, and U. Zwick. Listing triangles. In *Proceedings of 41st International Colloquium on Automata, Languages and Programming (ICALP)*, 2014. 2, 4.1.1, 4.1.1

[15] B. Bollobás. *Random graphs*, volume 73 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, second edition, 2001. 3

[16] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(9):1124–1137, 2004. 5.1

[17] G. Buehrer and K. Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *WSDM*, pages 95–106. ACM, 2008. 1

[18] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *APPROX*, 2000. (document), 1, 2, 4.1, 4.2, 4.2, 5.2

[19] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick.

Reachability and distance queries via 2-hop labels. In *SODA*, 2002. 1

[20] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008. 4.3

[21] A. Epasto and S. Lattanzi and M. Sozio. Efficient Densest Subgraph Computation in Evolving Graphs. *WWW'15* (to appear), 2015. 2

[22] U. Feige, G. Kortsarz, and D. Peleg. The dense k-subgraph problem. *Algorithmica*, 29(3), 2001. 2

[23] I. Finocchi, M. Finocchi, and E. G. Fusco. Counting small cliques in mapreduce. *arXiv preprint arXiv:1403.0734*, 2014. 2

[24] E. Fratkin, B. T. Naughton, D. L. Brutlag, and S. Batzoglou. Motifcut: regulatory motifs finding with maximum density subgraphs. *Bioinformatics*, 22(14):e150–e157, 2006. 1

[25] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM Journal on Computing*, 18(1):30–55, 1989. 1, 2, 4.1.1

[26] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB*, 2005. 1

[27] A. Gionis, F. Junqueira, V. Leroy, M. Serafini, and I. Weber. Piggybacking on social networks. *Proc. VLDB Endow.*, 6(6):409–420, Apr. 2013. 1

[28] A. V. Goldberg. Finding a maximum density subgraph. Technical report, University of California at Berkeley, 1984. 1, 2, 4.1.1, 5.2

[29] M. T. Goodrich and P. Pszona. External-memory network analysis algorithms for naturally sparse graphs. In *Algorithms–ESA 2011*, pages 664–676. Springer, 2011. 4.3

[30] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*. Springer, Berlin, 1988. 4.1.2

[31] D. Gusfield. Computing the strength of a graph. *SIAM Journal on Computing*, 20(4):639–654, 1991. 4.1.1, 4.4

[32] J. Hastad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182(1), 1999. 1

[33] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978. 2, 4.1.1, 4.1.1

[34] S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM (JACM)*, 48(4):761–777, 2001. 4.1.2

[35] M. Jha, C. Seshadhri, and A. Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 589–597. ACM, 2013. 2

[36] J. Jiang, M. Mitzenmacher, and J. Thaler. Parallel peeling algorithms. *arXiv preprint arXiv:1302.7014*, 2013. 4.2

[37] R. Jin, Y. Xiang, N. Ruan, and D. Fuhry. 3-hop: a high-compression indexing scheme for reachability query. In *SIGMOD*, 2009. 1

[38] R. Kannan and V. Vinay. Analyzing the structure of large graphs, 1999. 2

[39] H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for mapreduce. In *SODA*, pages 938–948. Society for Industrial and Applied Mathematics, 2010. 1

[40] S. Khuller and B. Saha. On finding dense subgraphs. In *ICALP*, 2009. 1, 2, 4.2

[41] M. N. Kolountzakis, G. L. Miller, R. Peng, and T. C. E. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Internet Mathematics*, 8(1-2):161–185, 2012. 2

[42] L. Lovász. Submodular functions and convexity. In *Mathematical Programming The State of the Art*, pages 235–257. Springer, 1983. 4.1.2

[43] J. B. Orlin. A faster strongly polynomial time algorithm for submodular function minimization. *Mathematical Programming*, 118(2):237–251, 2009. 2, 4.1.2

[44] R. Pagh and C.E. Tsourakakis. Colorful triangle counting and a mapreduce implementation. *Information Processing Letters*, 112(7):277–281, 2012. 2, 7

[45] A. D. Sarma, A. Lall, D. Nanongkai, and A. Trehan. Dense subgraphs on dynamic networks. In *Distributed Computing*, pages 151–165. Springer, 2012. 2

[46] A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B*, 80(2):346–355, 2000. 4.1.2

[47] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *KDD*, pages 939–948. ACM, 2010. 6

[48] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *WWW*, pages 607–614. ACM, 2011. 1, 4.3

[49] C.E. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. A.Tsiarli. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *KDD*, pages 104–112. ACM, 2013. 1, 2, 6

[50] C. E. Tsourakakis. A novel approach to finding near-cliques: The triangle-densest subgraph problem. *arXiv preprint arXiv:1405.1477*, 2014. 4.1

[51] C. E. Tsourakakis, M. N. Kolountzakis, and G. L. Miller. Triangle sparsifiers. *J. Graph Algorithms Appl.*, 15(6):703–726, 2011. 2

[52] N. Wang, J. Zhang, K.-L. Tan, and A. K. Tung. On triangulation-based dense neighborhood graph discovery. *Proceedings of the VLDB Endowment*, 4(2):58–68, 2010. 2

[53] V. V. Williams. Multiplying matrices faster than coppersmith-winograd. In *STOC*, pages 887–898. ACM, 2012. 2, 4.1.1, 4.1.2

[54] Y. Zhang and S. Parthasarathy. Extracting analyzing and visualizing triangle k-core motifs within networks. In *ICDE*, pages 1049–1060. IEEE, 2012. 2