Large Scale Density-friendly Graph Decomposition via Convex Programming*

Maximilien Danisch
LTCI, Télécom ParisTech,
Université Paris-Saclay,
75013, Paris, France
danisch@telecom-paristech.fr

T-H. Hubert Chan†
Department of Computer Science, The University of Hong Kong, Hong Kong, China
hubert@cs.hku.hk

Mauro Sozio‡
LTCI, Télécom ParisTech,
Université Paris-Saclay,
75013, Paris, France
sozio@telecom-paristech.fr

ABSTRACT

Algorithms for finding dense regions in an input graph have proved to be effective tools in graph mining and data analysis. Recently, Tatti and Gionis [WWW 2015] presented a novel graph decomposition (known as the locally-dense decomposition) that is similar to the well-known k-core decomposition, with the additional property that its components are arranged in order of their densities. Such a decomposition provides a valuable tool in graph mining. Unfortunately, their algorithm for computing the exact decomposition is based on a maximum-flow algorithm which cannot scale to massive graphs, while the approximate decomposition defined by the same authors misses several interesting properties. This calls for scalable algorithms for computing such a decomposition. In our work, we devise an efficient algorithm which is able to compute exact locally-dense decompositions in real-world graphs containing up to billions of edges. Moreover, we provide a new definition of approximate locally-dense decomposition which retains most of the properties of an exact decomposition, for which we devise an algorithm that can scale to real-world graphs containing up to tens of billions of edges. Our algorithm is based on the classic Frank-Wolfe algorithm which is similar to gradient descent and can be efficiently implemented in most of the modern architectures dealing with massive graphs. We provide a rigorous study of our algorithms and their convergence rates. We conduct an extensive experimental evaluation on multi-core architectures showing that our algorithms converge much faster in practice than their worst-case analysis. Our algorithm is even more efficient for the more specialized problem of computing a densest subgraph.

†This research was partially supported by the Hong Kong RGC under the grants 17202715 and 17217716.

‡This research was partially supported by French National Agency (ANR) under project FIELDS (ANR-15-CE23-0006) and by a Google Faculty Award.

*This research was partially supported by a grant from the PROCORE France-Hong Kong Joint Research Scheme sponsored by the Research Grants Council of Hong Kong and the Consulate General of France in Hong Kong under the project F-HKU702/16.

WWW 2017, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3038912.3052619

1. INTRODUCTION

Algorithms for finding dense regions in an input graph have proved to be valuable tools in graph mining with applications in biology [20], finance [16], web mining [21], as well as real-time story identification [3]. On the other hand, the well-known k-core decomposition stands out for its simplicity and its ability to unravel the structural organization of a graph. It has been successfully applied in many contexts such as speeding up algorithms [19, 33], finding best spreaders [27], drawing large graphs [2], bioinformatics [5], analyzing human brains [23] and team formation [10].

Recently, Tatti and Gionis [38] proposed a novel graph decomposition, known as the *locally-dense graph decomposition.* Such a decomposition boasts similar properties to the k-core decomposition with the additional property that its components are nested into one another, with inner components having larger density than outer ones. Moreover, the locally-dense decomposition contains all the so-called *locally-dense* subgraphs of the input graph. The locally-dense graph decomposition provides a valuable tool in graph mining.

Unfortunately, their algorithm for computing the exact decomposition does not scale to massive graphs, while the approximate decomposition defined by the same authors may not contain any non-trivial locally-dense subgraph.

In our work, we devise an efficient algorithm for computing exact locally-dense decompositions in massive graphs. Our main algorithm is based on a variant of the classic Frank-Wolfe algorithm that is similar to gradient descent and can be efficiently implemented in most of the modern architectures dealing with massive graphs. We provide a rigorous worst-case analysis of its convergence rate. We give a different definition of approximate decomposition than the one given in [38]. Our notion of approximation is stronger in the sense that it computes a non-trivial subset of all locally-dense subgraphs, while locally-dense subgraphs with very different densities will still be distinguished. We devise an efficient algorithm for computing an approximate graph decomposition that, for any $\epsilon > 0$, computes a $(1+\epsilon)$-approximation of the exact decomposition.

We conduct experimental evaluations on real-world graphs containing up to 25 billion edges, for which our main algorithm exhibits faster convergence rate than the worst-case analysis. In our experiments, we focus on multi-core archi-

tectures. However, our main algorithm can be efficiently implemented in other well known architectures such as MapReduce and Spark thanks to its simplicity and its fast rate of convergence on real-world graphs. Our work illustrates the potential of the Frank-Wolfe algorithm in large scale graph mining, which perhaps has not been fully exploited, yet.

Related work. Our paper is related to previous work on graph decompositions as well as finding dense subgraphs. We review some representative work on these two topics.

In addition to the k-core decomposition, other decompositions have been studied. The truss decomposition [13, 41] can be seen as a generalization of the core decomposition to triangles, since a k-truss is defined as a subgraph in which each edge is contained in at least $(k-2)$ triangles. The modular decomposition [14] is a decomposition of a graph into a hierarchy of subsets of vertices called modules. A module is defined in the following way: X is a module if, for each vertex $v \notin X$, either every member of X is a non-neighbor of v or every member of X is a neighbor of v. The nucleus decomposition [35] is a decomposition of a graph into a hierarchy of dense subgraphs. It is worth to mention the work in [24, 25], which shows an interesting connection between load balancing and the density-friendly decomposition.

Dense subgraphs detection has been widely studied [29]. Such a problem aims at finding a subgraph of a given input graph that maximizes some notion of density. The most common density notion employed in the literature is the average degree. Due to its popularity, the corresponding problem of finding a subgraph that maximizes the average degree has been commonly referred to as the densest-subgraph problem. The densest subgraph can be identified in polynomial time by solving a parametric maxflow problem [22], while a simple greedy algorithm based on k-core decomposition produces a $\frac{1}{2}$-approximation in linear time [12]. The densest-subgraph problem has also been studied in streaming and dynamic graphs [6, 18, 8]. Other notions of density have also been investigated such as the minimum degree density [37] or density based on the number of triangles [42, 34] which can be solved in polynomial time. Other definition of density leading to NP-hard problems have also been investigated such as [4, 31] where an upper bound on the number of nodes in the graph is enforced as well as quasi-clique detection [1, 39].

The rest of the paper is organized as follows. In Section 2 we introduce the notations, definitions and basic theorems necessary for the understanding of our work. In Section 3, we present our algorithms for the computation of the exact locally-dense decomposition as well as its approximation. In Section 4, we present our theoretical analysis of the problems and of our algorithms. We then evaluate the performances of our algorithms against the state of the art in Section 5.

2. PRELIMINARIES

We consider a weighted undirected graph $G = (V_G, E_G, w)$, where $w : E_G \to \mathbb{R}^+$ is a weight function on the edges of G. We denote by n and m the number of nodes and edges, respectively. We allow G to contain self-loops, i.e., there might be an edge consisting of one single node. We interpret each edge $e \in E_G$ as a subset $e \subset V_G$ of nodes. For ease of exposition, in this paper, we concentrate on the case that each edge e contains at most 2 nodes, but our approach can be readily generalized to hypergraphs (with arbitrary edge cardinality). The density of a subgraph $H = (V_H, E_H)$ in G is defined as $\rho_G(H) := \frac{w(E_H)}{|V_H|}$, where $w(E_H)$ is the sum of the weights of the edges with all nodes in E_H. A subgraph H of G is a *densest subgraph* if and only if H has maximum density among all subgraphs of G. Observe that a densest subgraph is a subgraph induced by some subset of nodes.

For a non-empty $S \subseteq V_G$, we define the induced subgraph $G[S] := (S, E_G(S), w)$ by S in G, where $E_G(S) := \{e \in E_G : e \subseteq S\}$ is the set of edges in G contained in S. For convenience of notation, given a set of nodes $S \subseteq V$ we define $\rho_G(S)$ to be the density of the graph $G[S]$. A set S is called a *densest subset* in G if it induces a densest subgraph in G. We shall drop the subscripts G and H when the graph in question are clear from the context. The following fact is standard for normal graphs, and a proof for hypergraphs can be readily generalized by [7, Lemma 4.1].

Fact 2.1 *The* maximal densest subgraph *of G is* unique *and contains all densest subgraphs of G.*

The definition of quotient graph is instrumental in the definition of our graph decomposition.

Definition 2.2 (Quotient Graph) *Given a weighted undirected graph $G = (V, E, w)$, and a subset $B \subseteq V$, the quotient graph of G with respect to B is a graph $G \setminus B = (\widehat{V}, \widehat{E}, \widehat{w})$, which is defined as follows.*

- $\widehat{V} := V \setminus B$.
- $\widehat{E} := \{e \cap \widehat{V} : e \in E, e \cap \widehat{V} \neq \emptyset\}$, *i.e., every edge $e \in E$ not contained in B contributes towards $\widehat{E}$.*
- *For $e' \in \widehat{E}$, $\widehat{w}(e') := \sum_{e \in E : e' = e \cap \widehat{V}} w(e)$.*

For ease of presentation, we give an alternative constructive definition for the locally-dense decomposition as follows.

Definition 2.3 (Diminishingly-Dense Decomposition) *Given a weighted undirected graph $G = (V, E, w)$, we define the diminishingly-dense decomposition $\mathcal{B}$ of G as the sequence $\emptyset = B_0 \subsetneq B_1 \subsetneq B_2 \subsetneq \cdots \subsetneq B_k = V$ as follows. Initially, we set $B_0 := \emptyset$ and $G_0 := G$.*

For $i \geq 1$, if $B_{i-1} = V$, the decomposition is fully defined. Otherwise, let $G_i := G_{i-1} \setminus B_{i-1} = (V_i, E_i, w_i)$ be the quotient graph of G_{i-1} with respect to B_{i-1}. Let S_i be the maximal densest subset in G_i (with respect to weight w_i and the corresponding density ρ_i). We define $B_i := B_{i-1} \cup S_i$.

For each $i = 1, \ldots, k$, we denote $r_i = \rho_i(S_i)$. Moreover, we define the maximal density *vector $r^G \in \mathbb{R}^V$ such that if $u \in S_i$, then $r^G(u) := r_i$.*

The motivation for the name of the decomposition is given by Lemma 2.7 and 2.8. Similarly we refer to the reverse of the sequence $B_k, B_{k-1}, \ldots, B_0$ as the *increasingly-dense decomposition*. We shall see in Section 2.1 that Definition 2.3 is equivalent to the locally-dense decomposition given in [38]. The main problem in this paper is as follows.

Problem Definition (*Diminishingly-Dense Decomposition Problem*). Given a graph $G = (V, E, w)$, find its diminishingly-dense decomposition $\emptyset = B_0 \subsetneq B_1 \subsetneq B_2 \subsetneq \cdots \subsetneq B_k = V$.

In addition to finding the exact decomposition, we also consider approximate diminishingly-dense decomposition. To describe the approximate version formally, we use the notion *additional density* [38].

Definition 2.4 (Additional Density) *Given subsets T and non-empty S such that S is not contained in T, the additional density*[1] *of S with respect to T is defined as* $\rho(S|T) := \frac{w(E(S\cup T)\setminus E(T))}{|S\setminus T|}$.

Definition 2.5 (Approximate Decomposition) *Given a graph $G = (V, E, w)$, suppose $\emptyset = B_0 \subsetneq B_1 \subsetneq B_2 \subsetneq \cdots \subsetneq B_k = V$ is the diminishingly-dense decomposition of G. Then, for $\epsilon \geq 0$, a chain of subsets $\emptyset = C_0 \subsetneq C_1 \subsetneq C_2 \subsetneq \cdots \subsetneq C_l = V$, where $l \leq k$, is an ϵ-approximate diminishingly-dense decomposition, if the following holds.*

1. *$\{C_i\}$ is a subsequence of $\{B_i\}$, and*
2. *for each $1 \leq i \leq l$, for all T such that $C_{i-1} \subsetneq T$, $\rho(T|C_{i-1}) \leq (1+\epsilon)\cdot\rho(C_i|C_{i-1})$.*

Observing the following fact, one can show that the only 0-approximate decomposition is exactly the diminishingly-dense decomposition.

Fact 2.6 (Relating Additional Densities) *Suppose the quotient graph $G\setminus B = (\widehat{V}, \widehat{E}, \widehat{w})$ has the corresponding density function $\widehat{\rho}$. Then, for disjoint subsets X and Y of $\widehat{V}$, we have $\rho(X|B\cup Y) = \widehat{\rho}(X|Y)$.*

2.1 Properties of Decompositions

An important property of the decomposition in Definition 2.3 is that the additional densities along the chain is strictly decreasing.

Lemma 2.7 (Diminishing Additional Densities) *In the diminishingly-dense decomposition in Definition 2.3, if $B_i \subsetneq V$, then $r_i > r_{i+1}$.*

Proof. Suppose on the contrary $r_i \leq r_{i+1}$, i.e., there exists a non-empty subset $X \subseteq V_{i+1}$ such that $\rho_{i+1}(X) = r_{i+1}$. Recall that S_i is the maximal densest subset in G_i. Suppose we now consider the density of $S_i \cup X$ in G_i. Observe that the additional contribution of edge weights due to X is $\rho_{i+1}(X)\cdot|X|$. Hence, we have $w_i(S_i\cup X) = r_i\cdot|S_i| + r_{i+1}\cdot|X| \geq r_i\cdot|S_i\cup X|$. Therefore, $\rho_i(S_i\cup X)\geq r_i$, contradicting the maximality of S_i. □

Lemma 2.8 shows that the diminishingly-dense decomposition consists of nested subgraphs with outer subgraph having lower densities than inner subgraphs.

Lemma 2.8 (Diminishing Densities) *In the diminishingly-dense decomposition in Definition 2.3, if $B_i \subsetneq V$, then $\rho(B_i) > \rho(B_{i+1})$.*

Proof. We have $\rho(B_i) = \sum_{j=0}^{j=i-1} \frac{|B_{j+1}\setminus B_j|}{|B_i|}\cdot r_{j+1} > r_{i+1}$, where the latter inequality follows from the fact that $\rho(B_i)$ can be expressed as a weighted average of quantities all strictly larger than r_{i+1}. On the other hand, $\rho(B_{i+1}) = \frac{\rho(B_i)\cdot|B_i| + r_{i+1}\cdot|B_{i+1}\setminus B_i|}{|B_{i+1}|} < \rho(B_i)$ (as $r_{i+1} < \rho(B_i)$). □

We next show that our graph decomposition in Definition 2.3 is equivalent to the locally-dense decomposition defined in [38].

[1] In [38], they use $d(S,T)$ to denote $\rho(S|T)$.

Definition 2.9 (Locally Dense Subset) *A non-empty $W \subseteq V$ is locally dense if for all $X \subseteq W$ and non-empty Y disjoint from W, $\rho(X|W\setminus X) > \rho(Y|W)$.*

It is shown in [38] that for any two locally dense S and T, then either $S \subseteq T$ or $T \subseteq S$. The same proof generalizes readily to hypergraphs. Therefore, the locally-dense decomposition can be defined as follows.

Definition 2.10 *Given a graph G, its locally-dense decomposition is the (unique) maximal chain of locally dense subsets: $B_1 \subsetneq B_2 \subsetneq \cdots \subsetneq B_k = V$.*

The purpose of the following lemmas is to show that the diminishingly-dense decomposition gives a maximal chain of locally dense subsets, which must also be the (unique) locally-dense decomposition. The proofs repeatedly apply the definition of additional density. Because of limited space, the proof of Lemma 2.11 is omitted.

Lemma 2.11 (Diminishingly-Dense Decomposition is Locally Dense) *Each B_i in the diminishingly-dense decomposition is locally dense.*

Lemma 2.12 (The Diminishingly-Dense Decomposition is a Maximal Locally-Dense Chain) *In the diminishingly-dense decomposition, for all $B_i \subsetneq X \subsetneq B_{i+1}$, the subset X is not locally dense.*

Proof. Suppose the contrary is true. Then, consider the non-empty sets $S := X\setminus B_i \subsetneq S_{i+1}$ and $Y := B_{i+1}\setminus X \subsetneq S_{i+1}$. We next consider the weight of the edges E' in G_{i+1} that are totally contained in S_{i+1}. Then, we have $w_{i+1}(E') = \rho_{i+1}\cdot|S_{i+1}|$. We partition E' into E_1 and E_2, where E_1 is the set of edges totally contained in S and $E_2 := E'\setminus E_1$.

We have $w_{i+1}(E_1) = \rho_{i+1}(S)\cdot|S| = \rho(S|B_i)\cdot|S|$, where the last equality follows from Fact 2.6.

On the other hand, $w_{i+1}(E_2) = \rho(Y|X)\cdot|Y|$. Hence, it follows that $\rho(S|B_i)\cdot|S| + \rho(Y|X)\cdot|Y| = \rho_{i+1}\cdot(|S|+|Y|)$. Since S_{i+1} is the maximal densest subset in G_{i+1}, it follows that $\rho(S|B_i)\leq r_{i+1}$, which implies that $\rho(Y|X)\geq r_{i+1}$.

Therefore, we have $\rho(S|X\setminus S) = \rho(S|B_i) \leq \rho(Y|X)$, which means X is not locally dense. □

Corollary 2.13 (Equivalence of Decompositions) *The diminishingly-dense decomposition is equivalent to the locally-dense decomposition. Hence, from now on, we simply refer to either notion as the (exact) decomposition.*

3. ALGORITHMS

The main idea of our algorithms is to compute (or approximate) the maximal density vector r^G in Definition 2.3, from which the decomposition can be recovered by sorting the nodes in non-increasing order on the coordinates of r^G.

Auxiliary Vector. We give a useful view on the density vector. We can imagine that each edge $e \in E$ distributes its weight w_e among the nodes contained in e. This can be captured by an auxiliary vector α in $\mathcal{D}(G) := \{\alpha \in \mathbb{R}_+^{\sum_{e\in E}|e|} : \forall e\in E, \sum_{u\in e}\alpha_u^e = w_e\}$, and in particular, α_u^e is the weight received by node u from edge e. Indeed, our algorithms maintain such an auxiliary vector and enforce the following invariant relating the density vector $r \in \mathbb{R}^V$ with the auxiliary vector in $\alpha \in \mathcal{D}(G)$.

Invariant Pair. We say that $(r \in \mathbb{R}_+^V, \alpha \in \mathcal{D}(G))$ is an invariant pair, if for all $u \in V$, $r(u) = \sum_{e \in E: u \in e} \alpha_u^e$.

We give our subroutines and their intuition in this section. Their analysis will be given in Section 4.

3.1 Frank-Wolfe Based Algorithm

The intuition is that in a densest subset S in graph G, it is possible for each edge $e \in E(S)$ to distribute its weight among its own nodes such that the total weight received by each node in S is exactly the density $\rho(S) = \frac{w(E(S))}{|S|}$. Moreover, in calculating the density of S, edges that are cut by S do not contribute their weights to any node in S.

This suggests a general framework for an iterative algorithm, which maintains the invariant between the density vector $r \in \mathbb{R}_+^V$ and the auxiliary vector $\alpha \in \mathcal{D}(G)$. In each iteration of Algorithm 1, each edge $e \in E$ attempts to distribute its weight w_e towards a node $x \in e$ whose current density $r(x)$ is minimum among the nodes in e. The algorithm is highly parallelizable over work performed per edge and per node.

Algorithm 1 Frank-Wolfe Based Algorithm

function FRANK-WOLFE($G = (V, E, w)$, $T \in \mathbb{Z}_+$)
  for each $e = uv$ in E in parallel **do**
    $\alpha_u^{e\,(0)}, \alpha_v^{e\,(0)} \leftarrow \frac{w_e}{2}$
  for each $u \in V$ in parallel **do**
    $r^{(0)}(u) \leftarrow \sum_{e \in E: u \in e} \alpha_u^{e\,(0)}$
  for each iteration $t = 1, \ldots, T$ **do**
    $\gamma_t \leftarrow \frac{2}{t+2}$
    for each e in E in parallel **do**
      $x \leftarrow \arg\min_{v \in e} r^{(t-1)}(v)$
      for each $u \in e$ **do**
        $\widehat{\alpha}_u^e \leftarrow w_e$, if $u = x$ and 0 otherwise.
    $\alpha^{(t)} \leftarrow (1 - \gamma_t) \cdot \alpha^{(t-1)} + \gamma_t \cdot \widehat{\alpha}$
    for each $u \in V$ in parallel **do**
      $r^{(t)}(u) \leftarrow \sum_{e \in E: u \in e} \alpha_u^{e\,(t)}$
  return $(\alpha^{(t)}, r^{(t)})$

In Section 4, we will show that Algorithm 1 is in fact a variant of the Frank-Wolfe algorithm described in [26], which converges to an optimum solution of a carefully defined constrained convex program. We will show that in an optimal solution, the density vector must be the maximal density vector r^G in Definition 2.3. However, the iterative algorithm might produce solutions that approach an optimal solution without ever exactly attaining it. Hence, we need other subroutines to recover the exact or an approximate decomposition when the current solution is "good" enough.

3.2 Tentative Graph Decomposition

Even though we derive theoretical convergence rates in Section 4, in practice various heuristics can help us to recover the decomposition faster. Our heuristic consists of computing a tentative decomposition starting from the density vector $r \in \mathbb{R}_+^V$ produced by the Frank-Wolfe based algorithm. The quality of such a decomposition is then verified in Sections 3.3 and 3.4. Such a tentative decomposition is computed as follows.

We first sort the nodes according to the density vector r: $r(u_1) \geq r(u_2) \geq \cdots \geq r(u_n)$. For each $1 \leq i \leq n$, define $y_i := \sum_{e \in E: i = \max\{j: u_j \in e\}} w_e$, i.e., the sum of the weights of edges that contain u_i as the node with the largest index. As observed in [11], the PAVA algorithm can be employed to compute a solution for the following problem.

For each $1 \leq j \leq n$, compute the maximum $m(j) \geq j$ such that $\frac{1}{m_j - j + 1} \sum_{k=j}^{m_j} y_k$ is maximized. The PAVA algorithm requires $O(n)$ time [40] in total. The $m(j)$'s computed by PAVA define a partition $[p_1, p_2 - 1], [p_2, p_3 - 1], \ldots, [p_l, p_{l+1}]$ of $[1, n]$, where $p_1 = 1$, $p_{l+1} = n$, while $p_{j+1} - 1 = m(p_j)$. Let S_j be the set containing all nodes u_s with $s \in [p_j, p_{j+1} - 1]$, $j = 1, \ldots, l$. From the partition $(S_1, S_2, \ldots, S_l)$, for each $1 \leq j \leq l$, a candidate subset $\widehat{B}_i := \cup_{j=1}^i S_j$ can be formed to be verified in the next step. We refer to this subroutine as TRYDECOMPOSE($G = (V, E, w), r \in \mathbb{R}_+^V$).

3.3 Verification of the Graph Decomposition

We next give a subroutine that verifies whether a candidate subset B appears in the chain of the exact decomposition of the given graph G. Our subroutine is based on a characterization in terms of stable subsets defined as follows.

Definition 3.1 (Stable Subset) *A non-empty subset $B \subseteq V$ is stable with respect to the invariant pair $(r \in \mathbb{R}_+^V, \alpha \in \mathcal{D}(G))$, if the following conditions hold.*

(a) For all $u \in B$ and $v \notin B$, $r(u) > r(v)$.

(b) For all $e \in E$ such that e intersects both B and $V \setminus B$, $\alpha_u^e = 0, \forall u \in e \cap B$.

In Lemma 4.11 of Section 4, we prove that if B is stable with respect to some pair (r, α), then B appears in the chain of the exact decomposition. This allows us to design the verification subroutine illustrated in Algorithm 2. Then, given a sequence $B_1 \subseteq B_2, \ldots, \subseteq B_l$ we can obtain a subsequence of the exact decomposition as follows. Starting from B_1 we verify by means of Algorithm 2 whether such a subset is stable. If this is the case, B_1 is included in the subsequence, while for each edge $e = uv$ with u in B_1 and v in $V \setminus B_1$, we set $\alpha_u^e = 0$ and $\alpha_v^e = w_e$. Otherwise, B_1 is not included in the subsequence. We iterate those steps for the rest of the B_i's while processing them in the order given by the sequence. We denote this subroutine by EXTRACTSTABLESUBSETS($G, B_1, \ldots, B_l, r, \alpha$).

Algorithm 2 Verify Stable Subsets

function ISSTABLE($G = (V, E, w)$, $B \subseteq V$, $\alpha \in \mathcal{D}(G)$)
  $\widehat{\alpha} \leftarrow \alpha$
  for all $e = uv$ such that $u \in B, v \in V \setminus B$ **do**
    $\widehat{\alpha}_u^e \leftarrow 0$, $\widehat{\alpha}_v^e \leftarrow w_e$
  for all $u \in V$ **do**
    $\widehat{r}(u) \leftarrow \sum_{e \in E: u \in e} \widehat{\alpha}_u^e$
  if $\forall u \in B, v \in V \setminus B, \widehat{r}(u) > \widehat{r}(v)$ **then**
    return SUCCESS
  return FAIL

3.4 Estimating the Decomposition Error

Given a graph decomposition $B_1, \ldots, B_l$, where each of the B_i's is a stable subset, we develop an algorithm for computing an upper bound on the error of the decomposition (with respect to the exact decomposition), as defined in Definition 2.5. Given a graph $G = (V, E, w)$ and an invariant pair $(r \in \mathbb{R}_+^V, \alpha \in \mathcal{D}(G))$, Fact 4.1 states that the maximum coordinate $r_{\max} := \max_{u \in B_i \setminus B_{i-1}} r(u)$ gives an

upper bound on the densest subgraph in $G \setminus B_{i-1}$. This suggests the following subroutine. For each B_i, $i = 1, \ldots, l$ let $r_i^{\max}$ be the maximum value in $\{r(v) | v \in B_i \setminus B_{i-1}\}$. The maximum value $\frac{r_i^{\max}}{\rho_{G \setminus B_{i-1}}(B_i)} - 1$ among all $i = 1, \ldots, l$ gives then an upper bound on the error of the approximate decomposition. We refer to this subroutine as EstimateError$(G, B_1, \ldots, B_l, r \in \mathbb{R}_+)$.

Such a subroutine is used as stopping condition to determine whether the current decomposition satisfies the required approximation guarantee.

3.5 Approximate and Exact Decomposition

We now have all the ingredients to compute both an approximate and an exact graph decomposition. The algorithm for computing an approximate decomposition receives in input a parameter $\epsilon > 0$ specifying the required approximation guarantee. Our Frank-Wolfe based algorithm (Algorithm 1) is then executed for T iterations, where T is specified in input. Then in turn TryDecompose and ExtractStableSubsets are executed, so as to compute a graph decomposition of the input graph. The error of the approximate decomposition is then evaluated by means of EstimateError. If such an error is small enough the algorithm terminates, otherwise, the previous steps are iterated until the required approximation guarantee is obtained. An exact decomposition can then be obtained by running the maximum flow algorithm presented in [38] in parallel for each locally dense subgraph in the approximate decomposition [2]. We observe that the former steps are crucial when computing an exact decomposition, in that, they allow a parallel computation of the maximum flow algorithm on relatively small subgraphs of the input graph. A pseudocode of our algorithm is shown in Algorithm 3.

Algorithm 3 Approximate/Exact Decomposition

```
Input: G = (V, E, w), ε ∈ R_+, T ∈ Z_+
Output: The exact/approximate decomposition of G
repeat
    (α, r) ← Frank-Wolfe(G, T)
    B ← TryDecompose (G, r)
    B ← ExtractStableSubsets (G, B, r, α)
    δ ← EstimateError (G, B, r)
until δ > ε
if the exact decomposition is required then
    B ← further decomposition of B by running the max. flow algorithm in [38] in parallel for each set B_{i+1} \ B_i.
return B
```

4. ANALYSIS

We analyze the subroutines given in Section 3 and prove their correctness. Our invariant pair (r, α), where $r \in \mathbb{R}_+^V$ and $\alpha \in \mathcal{D}(G) := \{\alpha \in \mathbb{R}_+^{\sum_{e \in E} |e|} : \forall e \in E, \sum_{u \in e} \alpha_u^e = w_e\}$, is inspired from the Charikar's LP relaxation for densest subgraphs [12].

[2]More precisely, the maximum flow algorithm proposed in [38] must be adapted so as to deal with self-loops, which does not pose any particular issue.

$$\mathsf{LP}(G) \qquad \max \sum_{e \in E} w_e x_e$$
$$\text{s.t.} \quad x_e \le y_u, \qquad \forall u \in e$$
$$\sum_{u \in V} y_u = 1,$$
$$x_e, y_u \ge 0, \qquad \forall u \in V, e \in E$$

The dual of $\mathsf{LP}(G)$ can be formulated as

$$\mathsf{DP}(G) \qquad \min \quad \rho$$
$$\text{s.t.} \quad \rho \ge \sum_{e: u \in e} \alpha_u^e, \qquad \forall u \in V$$
$$\sum_{u \in e} \alpha_u^e \ge w_e, \qquad \forall e \in E$$
$$\alpha_u^e \ge 0, \qquad \forall u \in e \in E$$

Then, LP duality gives the following fact.

Fact 4.1 *Suppose (r, α) is an invariant pair for the graph $G = (V, E, w)$. Then, the maximum coordinate of r gives an upper bound of the maximum density of a subset in V.*

Convex Program. The intuition of our subroutines is that each edge $e \in E$ tries to distribute its weight among the nodes it contains such that the total weights received by the nodes are as even as possible. This suggests that we consider the objective function $Q_G(\alpha) := \sum_{u \in V} r(u)^2$, where (r, α) is an invariant pair, i.e., $r(u) = \sum_{e \in E: u \in e} \alpha_u^e$. Define the convex program $\mathsf{CP}(G) := \min\{Q_G(\alpha) : \alpha \in \mathcal{D}(G)\}$.

We analyze the correctness and the convergence rate of our algorithms in Section 3 by showing the following.

1. If $\alpha \in \mathcal{D}(G)$ is an optimum solution for $\mathsf{CP}(G)$, then the density vector $r \in \mathbb{R}_+^V$ induced from the invariant is exactly the maximal density vector r^G from Definition 2.3.
2. Our iterative Algorithm 1 is essentially the variant of Frank-Wolfe algorithm described in [26] applied to $\mathsf{CP}(G)$. Hence, theoretical convergence rates can readily be deduced.
3. Stable subsets correspond to subsets that appear in the exact decomposition.

4.1 Properties of an Optimal Solution of $\mathsf{CP}(G)$

The collection of the level sets of a vector $r \in \mathbb{R}^V$ is defined as $\{S_\rho : \exists u \in V, r(u) = \rho\}$, where each level set is $S_\rho := \{u \in V : r(u) \ge \rho\}$.

Lemma 4.2 (Stable Level Sets) *Suppose an optimal solution α of $\mathsf{CP}(G)$ induces the density vector $r \in \mathbb{R}_+^V$ via the invariant. Then, each level set of r is stable with respect to α.*

Proof. It suffices to prove that if there exists $u, v \in e \in E$ such that $r(u) > r(v)$, then $\alpha_u^e = 0$. Otherwise, there exists $\epsilon > 0$ such that we could decrease α_u^e by ϵ and increase α_v^e by ϵ to strictly decrease the objective function. □

Lemma 4.3 (Uniform Stable Subset) *Suppose that a non-empty $S \subseteq V$ is stable with respect to some $\alpha \in \mathcal{D}(G)$. Suppose further that there is some value $\rho \in \mathbb{R}$ such that for*

all $u \in S$, $r^\alpha(u) = \rho$. Then, S is the maximal densest subset in G and has density ρ.

PROOF. Under the feasible dual α, the nodes in S receive the maximum r^α values, since S is stable. Hence, by definition of $\mathsf{DP}(G)$, the objective value of α is ρ.

Moreover, since S is stable, the weight r^α received by nodes in S all comes from edges in $E(S)$. Therefore, it follows that the density $\rho(S) = \frac{w(E(S))}{|S|} = \rho$. Since the subset S corresponds to a feasible primal solution in $\mathsf{LP}(G)$ with objective value $\rho(S) = \rho$, it follows that ρ is the optimal value for both $\mathsf{LP}(G)$ and $\mathsf{DP}(G)$. Hence, S is a densest subset in G, and it suffices to prove that it is the maximal densest subset.

Suppose there exists a non-empty B disjoint from S such that $S \cup B$ is also a densest subset. Consider the quotient graph $\widehat{G} := G \setminus S$ with density $\widehat{\rho}$. Since $\rho(S \cup B) = \rho$, we have $\rho(B|S) = \widehat{\rho}(B) = \rho$. This means $\mathsf{LP}(\widehat{G})$ has a feasible solution with objective value at least ρ.

On the other hand, since S is stable with respect to α, all edges that intersect both S and $V \setminus S$ have all their weights distributed to $V \setminus S$. Hence, $\alpha \in \mathcal{D}(G)$ naturally induces $\widehat{\alpha} \in \mathcal{D}(\widehat{G})$ such that for all $u \in V \setminus S$, $r^\alpha(u) = r^{\widehat{\alpha}}(u) < \rho$. This means $\mathsf{DP}(\widehat{G})$ has a feasible dual with objective value strictly less than ρ. This contradicts weak duality, and hence we conclude that S must be the maximal densest subset in G. □

Lemmas 4.2 and 4.3 imply that an optimal solution α induces a density vector r whose level set corresponding to the maximum coordinate is the maximal densest subset in G. By an induction argument that applies Lemmas 4.2 and 4.3 repeatedly to the sequence of quotient graphs, we can obtain the following corollary.

Corollary 4.4 (Exact Decomposition) *Suppose an optimal solution α of $\mathsf{CP}(G)$ induces the density vector $r \in \mathbb{R}_+^V$ via the invariant. Then, the level sets of r give the exact decomposition of G, and r agrees with the local-maximal densities r^G in Definition 2.3.*

4.2 Frank-Wolfe Algorithm

We summarize the Frank-Wolfe algorithm described in [26], which solves a convex program $\min Q(\alpha)$ subject to $\alpha \in \mathcal{D}$, where Q is twice differentiable and $\mathcal{D}$ is a compact convex set in some Euclidean space. The algorithm is an iterative method similar to gradient descent, where we denote ∇Q as the gradient of Q.

Algorithm 4 Frank-Wolfe Algorithm on function Q and feasible set $\mathcal{D}$

1: **function** FRANK-WOLFE(Q,$\mathcal{D}$)
2: Set initial $\alpha^{(0)} \in \mathcal{D}$ arbitrarily.
3: **for** each iteration $t = 1, \ldots, T$ **do**
4: $\gamma_t \leftarrow \frac{2}{t+2}$
5: $\widehat{\alpha} \leftarrow \arg\min_{\alpha \in \mathcal{D}} \langle \alpha, \nabla Q(\alpha^{(t-1)}) \rangle$
6: $\alpha^{(t)} \leftarrow (1 - \gamma_t) \cdot \alpha^{(t-1)} + \gamma_t \cdot \widehat{\alpha}$
7: **return** $\alpha^{(t)}$

The following lemma shows that our iterative Algorithm 1 is actually a realization of Algorithm 4 applied to our objective function Q_G and feasible set $\mathcal{D}(G)$.

Lemma 4.5 *Lines (8) to (11) of Algorithm 1 implement line (5) in Algorithm 4.*

PROOF. Line (5) in Algorithm 4 solves the following problem. Given $\alpha \in \mathcal{D}(G)$, find $\widehat{\alpha} \in \mathcal{D}$ to minimize $\langle \widehat{\alpha}, \nabla Q(\alpha) \rangle = 2 \sum_{e \in E} \sum_{u \in e} \widehat{\alpha}_u^e \cdot r(u)$, where $r \in \mathbb{R}_+^V$ is induced by α via the invariant.

We can consider each edge $e \in E$ independently because the feasible set $\mathcal{D}(G)$ places a constraint $\sum_{u \in e} \widehat{\alpha}_u^e = w_e$ on each edge $e \in E$. Hence, to minimize $\sum_{u \in e} \widehat{\alpha}_u^e \cdot r(u)$, edge e should distribute its weight w_e entirely to a vertex $x \in e$ having minimum $r(x)$. This is achieved precisely by lines (8) to (11) of Algorithm 1. □

4.3 Rate of Convergence

After establishing that Algorithm 1 is a realization of Algorithm 4 in Lemma 4.5, we can use previous convergence analysis of the Frank-Wolfe algorithm [26].

The convergence rate of Algorithm 4 can be described by a constant $C_Q := \frac{1}{2}\mathsf{Diam}(\mathcal{D})^2 \sup_{\alpha \in \mathcal{D}} \|\nabla^2 Q(\alpha)\|_2$, where $\nabla^2 Q$ is the Hessian and $\|\cdot\|_2$ is the spectral norm of a matrix.

Theorem 4.6 (Convergence Rate of Frank-Wolfe [26]) *Suppose $\alpha^* \in \mathcal{D}$ is an optimal solution. Then, for all $t \geq 1$, $Q(\alpha^{(t)}) - Q(\alpha^*) \leq \frac{2C_Q}{t+2}$.*

Lemmas 4.7 and 4.8 are the results of some straightforward computation that we omit here.

Lemma 4.7 (Bounding C_Q) *For a graph $G = (V, E, w)$ with maximum node degree Δ (where the degree of a node is the number of edges that contain it), we have the corresponding $C_{Q_G} \leq 2\Delta \sum_{e \in E} w_e^2$.*

Lemma 4.8 (Error in r implies Error in Q) *Suppose $\alpha \in \mathcal{D}$ induces r such that $\epsilon := \|r - r^*\|_2$, where $r^* := r^G$ is induced by an optimal α^*. Then, $Q(\alpha) - Q(\alpha^*) \geq \epsilon^2$.*

The following corollary concludes the proof of the rate of convergence of Algorithm 1.

Corollary 4.9 (Convergence of Parallel Algorithm.) *Suppose Δ is the maximum degree of a node in G. In Algorithm 1, for $t > \frac{4\Delta \sum_{e \in E} w_e^2}{\epsilon^2}$, we have $\|r^{(t)} - r^G\|_2 \leq \epsilon$.*

4.4 Characterization of Locally-Dense Subsets

The following Lemma 4.10 (statement a) is needed to establish the correctness of ESTIMATEERROR in Section 3.4, while Lemma 4.11 proves the correctness of EXTRACTSTABLESUBSETS in Section 3.3.

Lemma 4.10 *Suppose $\alpha \in \mathcal{D}(G)$ is a feasible solution to $\mathsf{CP}(G)$. Then, the density vector r^G satisfies the following:*
(a) $\max_{u \in V} r^\alpha(u) \geq \max_{u \in V} r^G(u)$.
(b) $\min_{u \in V} r^\alpha(u) \leq \min_{u \in V} r^G(u)$.

PROOF. We have that $\max_{u \in V} r^G(u) = r_1$. From the fact that $\sum_{u \in B_1} \frac{r^\alpha(u)}{|B_1|} \geq \frac{\widehat{w}(\widehat{E}_1)}{|B_1|} = r_1$, statement (a) follows. Similarly, $\min_{u \in V} r^G(u) = r_k$ and $\sum_{u \in V \setminus B_{k-1}} \frac{r^\alpha(u)}{|V \setminus B_{k-1}|} \leq \frac{\widehat{w}(\widehat{E}_k)}{|V \setminus B_{k-1}|} = r_k$, from which statement (b) follows. □

networks	source	n	m
LiveJournal	[30]	4,036,538	34,681,189
Wikipedia	[32]	2,080,370	42,336,692
Orkut	[30]	3,072,627	117,185,083
Twitter	[28]	52,579,683	1,614,106,500
Friendster	[30]	124,836,180	1,806,067,135
gsh-2015	[9]	988,490,691	25,690,705,119

Table 1: Our set of large graphs.

Lemma 4.11 (Stable Subset is Locally-Dense) *Suppose non-empty $B \subseteq V$ is stable with respect to some feasible $\alpha \in \mathcal{D}(G)$. Then, B is locally-dense in G, i.e., B is one of the subsets in the diminishingly-dense decomposition.*

PROOF. Suppose that a subset B is stable with respect to some $\alpha \in \mathcal{D}(G)$. Because of Corollary 4.4, it suffices to prove that for all $u \in B$ and $v \notin B$, $r^G(u) > r^G(v)$.

Define the induced subgraph $G_1 := G[B]$ and the quotient graph $G_2 := G \setminus B$. Since B is stable with respect to α, α naturally induces $\alpha_1 \in \mathcal{D}(G_1)$ and $\alpha_2 \in \mathcal{D}(G_2)$, and we have for all $u \in B$ and $v \notin B$, $r^{\alpha_1}(u) > r^{\alpha_2}(v)$.

Suppose $\alpha_1^* \in \mathcal{D}(G_1)$ and $\alpha_2^* \in \mathcal{D}(G_2)$ are optimal solutions for $\mathsf{CP}(G_1)$ and $\mathsf{CP}(G_2)$, respectively. By Lemma 4.10, for all $u \in B$, $r^{\alpha_1^*}(u) \geq \min_{x \in B} r^{\alpha_1}(x)$, and for all $v \notin B$, $\max_{x \notin B} r^{\alpha_2}(x) \geq r^{\alpha_2^*}(v)$, which implies that $r^{\alpha_1^*}(u) > r^{\alpha_2^*}(v)$.

Therefore, α_1^* and α_2^* can be naturally combined to give an optimal solution $\alpha^* \in \mathcal{D}(G)$ for $\mathsf{CP}(G)$. Hence, we have for all $u \in B$ and $v \notin B$, $r^G(u) > r^G(v)$, as required. □

5. EXPERIMENTS

In our experimental evaluation, we consider six large real-world graphs containing up to 25 billion edges. Table 1 reports for each such a graph the statistics and the source from which the graph has been obtained. In all cases, directionality is ignored. We implemented our algorithms in C++, while employing openMP [15] for multi-threading[3]. For computing the maximum flow, we use the implementation of the Boykov-Kolmogorov available in the boost library [36] (as in [38]). This is done after our algorithm partitions the input graph in "sufficiently small" stable subsets. The maximum flow algorithm can then be run in parallel on each such stable subsets. We evaluate our algorithms against the algorithm developed in [38], while using their C++ implementation available on the web page of the authors. All codes are compiled using the -O3 optimization level. The experiments are carried out on a linux machine being equipped with 2 processors Intel Xeon CPU E5-2660 @ 2.60 GHz with 10 cores split in 2 threads each (a total of 40 threads), as well as 64G of RAM DDR4 2133 MHz. Unless otherwise specified, we employ 10 threads for running our Frank-Wolfe-based algorithm. In our experiments, we consider the asynchronous version of the Frank-Wolfe algorithm (where the α's can be modifed in any arbitrary order) which turns out to be more efficient in practice than the synchronous version. The fact that asynchronous versions are in practice more efficient than synchronous versions has been investigated for other related problems such as Belief Propagation [17]. We defer to future work a theoretical analysis of this fact for our problem.

For the largest network gsh-2015, we use another machine with very similar specifications. The only difference is that such a machine is equipped with 512G of RAM so that such a large graph can fit into main memory. Our goal in this case is to study the convergence of the algorithm and in particular the number of iterations required for our Frank-Wolfe-based algorithm to converge when such a large graph is considered. We defer to future work an evaluation of our algorithm when the input graph does not fit into main memory.

5.1 Rate of Convergence

We start by studying the rate of convergence of our Frank-Wolfe-based algorithm (Algorithm 1). In particular, we compare the rate of convergence of the algorithm on real-world graphs with the worst-case analysis provided in Corollary 4.9. This is illustrated in Figure 1 (left). Such a figure shows for each iteration t the L_2 norm of the difference between the maximal density vector r^G (computed at the end of Algorithm 3 (exact version)) and the vector $r^{(t)}$ computed at iteration t. Different datasets are depicted with different colors. Dashed lines represent the upper bound on the number of iterations provided in Corollary 4.9, while solid lines refer to the actual experiments on real-world graphs. Note that for gsh-2015 we only show the dashed lines as the exact decomposition could not be obtained for this graph.

We observe that our Frank-Wolfe-based algorithm converges much faster than its worst-case analysis. In the next sections, we show that the exact decomposition can already be obtained after approximately 500 iterations.

5.2 Approximate Decomposition

We evaluate the error in the approximate decomposition computed by our algorithm as a function of the number of iterations. As the exact decomposition is not known by our algorithm, we estimate the error in the approximate decomposition by means of ESTIMATEERROR in Section 3.4. Figure 1 (middle) illustrates this upper bound on the error ϵ, as defined in Definition 2.5, as a function of the number of iterations. As we can compute the exact decompositions for all our datasets except gsh-2015, it is possible for those datasets to measure the actual error on the decomposition as a function of the number of iterations, which is shown in Figure 1 (right). In particular, the latter figure shows that approximately 400 iterations suffice to achieve a 0.1-approximate decompositions on all datasets. It turns out, therefore, that the results shown in Figure 1 are rather conservative.

5.3 Comparison

In this section, we evaluate the running time of our algorithm as well as the error in the approximate decomposition, while comparing it against the algorithms presented in [38]. We run 500 iterations of our Frank-Wolfe based algorithm which in our experiments suffice to compute an exact decomposition for all our networks except gsh-2015. We report its total running time for all our networks. We then measure the error ϵ in the approximate decomposition (as defined in Definition 2.5) obtained after 500 iterations (without running the maximum flow algorithm) and report both the error and the running time. Table 2 shows the total running time of our approximation algorithm, our exact algorithm as well

[3]Our code is publicly available at `https://github.com/maxdan94/Density-Friendly`.

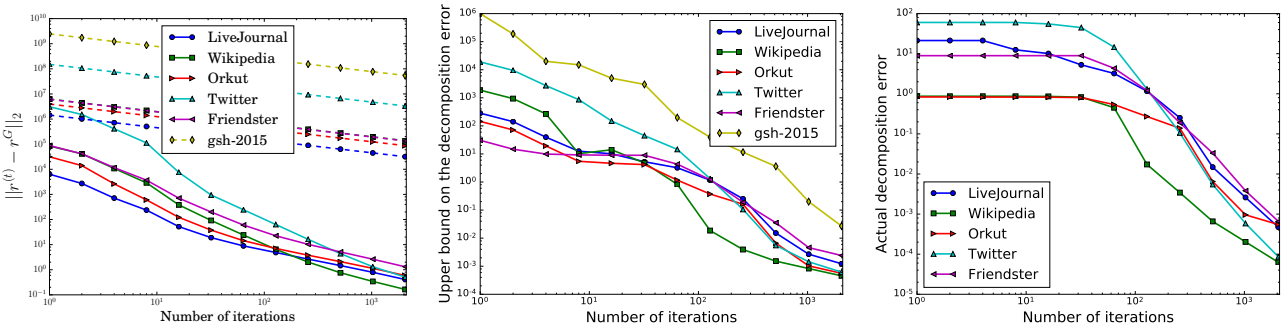

Figure 1: (left) $||r^{(t)} - r^G||_2$ as a function of the number iterations, dashed lines represent the upper bound provided in Corollary 4.9. (middle) Upper bound and (right) actual error of the approximate decomposition as a function of the number of iterations.

Networks	FW	TD	ESS	MF
LiveJournal	1m43s	14s	28s	10s
Wikipedia	1m36s	13s	13s	2s
Orkut	7m33s	47s	1m31s	2m47s
Twitter	1h44m34s	15m29s	1h1m27s	1h31m7s
Friendster	2h20m35s	16m47s	1h7m36s	1h54m39s

Table 3: Running time for each building block of our main algorithm. Our Frank-Wolfe based algorithm is run for 500 iterations.

as the exact and approximation algorithm developed in [38], denoted as TG15 (from the initials of the authors).

We can see that our exact algorithm is approximately four times as fast as TG15 on LiveJournal, Wikipedia and Orkut, while TG15 fails to run on the largest graphs. It turns out that TG15 requires a large amount of memory due to large number of variables which are required in the maximum flow computation. As a result, when TG15 is executed on Twitter or Friendster it runs out of memory after a few hours of computation. In contrast, our algorithm is able to compute an exact decomposition for these two networks within a few hours. Observe that one of the advantages of our exact algorithm lies on the fact that it computes a maximum flow algorithm on much smaller subgraphs (on the subgraphs of $G \setminus B_{i-1}$ induced by $B_i \setminus B_{i-1}$ computed in the previous steps of our algorithm). Table 2 reports also the error in the approximate decomposition computed by our algorithm without running the maximum flow algorithm. We can see that the error is very small, therefore, the computation of the maximum flow can be avoided if one is content with a good approximation of the exact decomposition. We also ran our approximation algorithm on gsh-2015. On this graph, with 2000 iterations, we obtain an approximate decomposition with a multiplicative error of $2.7 \cdot 10^{-2}$ within four days of computation.

Next, we evaluate how the different building blocks of our algorithm contribute to the total running time. Our Frank-Wolfe based algorithm is run for 500 iterations. Table 3 shows the running time of each of the steps in our algorithm for computing the exact decomposition, namely: (i) Frank-Wolfe based algorithm (FW for short) for which the total running time for 500 iterations is reported, (ii) the algorithm TryDecompose (TD) for tentative decomposition in Section 3.2, (iii) the algorithm ExtractStableSubsets (ESS) that extracts the stable sets from the aforementioned tentative decomposition, (iv) the algorithm which computes a maximum flow (MF) in each of the stable sets. The latter step is only used for computing the exact decomposition.

The results are shown in Table 3. It shows that the most computationally expensive steps of our main algorithm are the Frank-Wolfe based algorithm and the maximum flow algorithm, while the time required to compute a tentative decomposition is negligible. Therefore, in view of the results shown in Table 2, the running time of the main algorithm could be decreased by a couple of hours if one is content with a good approximation of the exact decomposition.

5.4 Degree of Parallelism

We study the degree of parallelism of our Frank-Wolfe based algorithm. We measure the speedup[4] as we increase the number of threads. Figure 2 (left) shows the running time (green curve) and the speedup (red curve) of the Frank-Wolfe based algorithm. We observe that a significant speedup is obtained (we obtain a speedup of 6 using 8 threads and a speedup of 17 using 32 threads) even tough it is not optimal. This speed up is not optimal as the step to compute the r values by summing the α values requires sharing locks among threads.

For our exact algorithm, we also use parallelization for the maximum-flow subroutine over the obtained independent blocks. However, the speedup is not so good as computing the maximum flow on the largest obtained independent block consumes almost all resources.

5.5 Densest subgraph

Here we show our experiments on finding only the densest subgraph and not the full decomposition. In this version of the algorithm, the first two steps (FW and TD) are the same, while the two last steps (ESS and MF) differ. Indeed, we can stop ESS as soon as the smallest stable set has been found, at this point we know that the densest subgraph is included in it. Therefore, we just need to run the maximum flow algorithm on this first found stable set.

We were able to find the exact densest subgraph following this straightforward modification using only 100 iterations

[4]The speedup with respect to k threads is defined as the running time using one thread divided by the running time using k threads.

Networks	approx.	TG approx.	exact	TG15
LiveJournal	2m35s (6.5e-3)	35s	2m45s	12m02s
Wikipedia	2m11s (2.2e-3)	36s	2m14s	7m07s
Orkut	10m21s (1.8e-3)	7m02s	13m08s	1h02m23s
Twitter	3h09m21s (5.8e-3)	1h35m26s	4h57m28s	-
Friendster	3h53m58s (1.3e-2)	1h43m54s	5h48m27s	-

Table 2: Overall running time comparison of our approximation algorithm (and approximation achieved), the approximation algorithm of TG, our exact algorithm and the exact algorithm of TG. We used 500 iterations for the gradient descent.

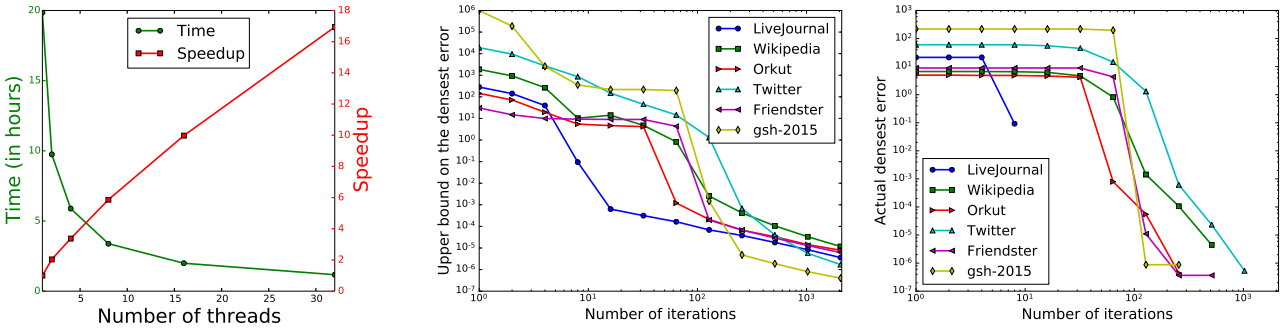

Figure 2: (left) Time and speedup versus number of threads on Friendster for 500 iterations for the iterations of the gradient descent. (middle) Upper bound and (right) actual multiplicative error on the densest subgraph as a function of the number of iterations.

on all our datasets including the largest one gsh-2015. On this largest network our algorithm took less than 10 hours of computation.

Figure 2 (left) shows the upper bound on the multiplicative error of the densest subgraph as a function of the number of iterations on all our graphs, while Figure 2 (right) shows the exact multiplicative error. As we can see, a very good approximation of the densest is obtained in a very small number of iterations (say after 300 iterations we obtain a 0.001-approximation on all datasets). In addition, we can see that on all datasets the approximation algorithm leads to the exact densest subgraph within 2048 iterations. In particular it took less than 16 iterations on LiveJournal and less than 512 iterations on gsh-2015.

6. CONCLUSION AND FUTURE WORK

We devised an algorithm based on the Frank-Wolfe gradient descent for computing the diminishingly-dense decomposition of very large graphs along with a strong theoretical analysis. Such a decomposition turns out to be equivalent to the locally-dense decomposition proposed in [38]. We showed that our algorithm is able to compute the exact decompositions for graphs of up 2 billion edges, as well as a good approximation of the exact decomposition in graphs of up to 25 billion edges. A densest subgraph can be computed even faster in all our datasets.

Future work includes an extensive experimental evaluation of our algorithms on hypergraphs for which all our theoretical results still hold. It would also be interesting to study from a theoretical point of view the asynchronous version of the Frank-Wolfe algorithm, which turns out to be more efficient in practice. Another interesting direction is to adapt our algorithm into the architectures for processing input data not fitting into main memory, such as Spark and MapReduce.

7. REFERENCES

[1] J. Abello, M. G. Resende, and S. Sudarsky. Massive quasi-clique detection. In *Latin American Symposium on Theoretical Informatics*, pages 598–612. Springer, 2002.

[2] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. Large scale networks fingerprinting and visualization using the k-core decomposition. In *Advances in neural information processing systems*, pages 41–50, 2005.

[3] A. Angel, N. Koudas, N. Sarkas, D. Srivastava, M. Svendsen, and S. Tirthapura. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *VLDB J.*, 23(2):175–199, 2014.

[4] Y. Asahiro, R. Hassin, and K. Iwama. Complexity of finding dense subgraphs. *Discrete Applied Mathematics*, 121(1):15–26, 2002.

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

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

[7] O. D. Balalau, F. Bonchi, T. Chan, F. Gullo, and M. Sozio. Finding subgraphs with maximum total density and limited overlap. In *WSDM*, pages 379–388, 2015.

[8] S. Bhattacharya, M. Henzinger, D. Nanongkai, and C. E. Tsourakakis. Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *STOC 2015*, pages 173–182, 2015.

[9] P. Boldi, A. Marino, M. Santini, and S. Vigna. BUbiNG: Massive crawling for the masses. In *WWW Companion*, pages 227–228, 2014.

[10] F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich. Core decomposition of uncertain graphs. In *SIGKDD*, pages 1316–1325, 2014.

[11] T. Calders, N. Dexters, J. J. Gillis, and B. Goethals. Mining frequent itemsets in a stream. *Information Systems*, 39:233–255, 2014.

[12] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *Approximation Algorithms for Combinatorial Optimization*, pages 84–95. Springer, 2000.

[13] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*, page 16, 2008.

[14] W. H. Cunningham. Decomposition of directed graphs. *SIAM Journal on Algebraic Discrete Methods*, 3(2):214–228, 1982.

[15] L. Dagum and R. Enon. Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.

[16] X. Du, R. Jin, L. Ding, V. E. Lee, and J. H. T. Jr. Migration motif: a spatial - temporal pattern mining approach for financial markets. In *SIGKDD*, pages 1135–1144, 2009.

[17] G. Elidan, I. McGraw, and D. Koller. Residual belief propagation: Informed scheduling for asynchronous message passing. In *UAI '06, Proceedings of the 22nd Conference in Uncertainty in Artificial Intelligence, Cambridge, MA, USA, July 13-16, 2006*, 2006.

[18] A. Epasto, S. Lattanzi, and M. Sozio. Efficient densest subgraph computation in evolving graphs. In *WWW*, pages 300–310, 2015.

[19] D. Eppstein and D. Strash. Listing all maximal cliques in large sparse real-world graphs. In *International Symposium on Experimental Algorithms*, pages 364–375. Springer, 2011.

[20] E. Fratkin, B. T. Naughton, D. L. Brutlag, and S. Batzoglou. Motifcut: regulatory motifs finding with maximum density subgraphs. In *Proceedings 14th International Conference on Intelligent Systems for Molecular Biology 2006, Fortaleza, Brazil, August 6-10, 2006*, pages 156–157, 2006.

[21] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *PVLDB*, pages 721–732, 2005.

[22] A. V. Goldberg. *Finding a maximum density subgraph.*

[23] P. Hagmann, L. Cammoun, X. Gigandet, R. Meuli, C. J. Honey, V. J. Wedeen, and O. Sporns. Mapping the structural core of human cerebral cortex. *PLoS Biol*, 6(7):e159, 2008.

[24] B. Hajek. Performance of global load balancing by local adjustment. *IEEE Transactions on Information Theory*, 36(6):1398–1414, 1990.

[25] B. Hajek et al. Balanced loads in infinite networks. *The Annals of Applied Probability*, 6(1):48–75, 1996.

[26] M. Jaggi. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *ICML*, pages 427–435, 2013.

[27] M. Kitsak, L. K. Gallos, S. Havlin, F. Liljeros, L. Muchnik, H. E. Stanley, and H. A. Makse. Identification of influential spreaders in complex networks. *Nature physics*, 6(11):888–893, 2010.

[28] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? In *WWW*, pages 591–600, 2010.

[29] V. E. Lee, N. Ruan, R. Jin, and C. Aggarwal. A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*, pages 303–336. Springer, 2010.

[30] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[31] M. Letsios, O. D. Balalau, M. Danisch, E. Orsini, and M. Sozio. Finding heaviest k-subgraphs and events in social media. In *The Sixth IEEE ICDM Workshop on Data Mining in Networks (DaMNet 2016), Barcelona Spain, 12 December 2016*, 2016.

[32] G. Palla, I. J. Farkas, P. Pollner, I. Derényi, and T. Vicsek. Fundamental statistical features and self-similar properties of tagged networks. *New Journal of Physics*, 10(12):123026, 2008.

[33] R. A. Rossi, D. F. Gleich, and A. H. Gebremedhin. Parallel maximum clique algorithms with applications to network analysis. *SIAM Journal on Scientific Computing*, 37(5):C589–C616, 2015.

[34] R. Samusevich, M. Danisch, and M. Sozio. Local triangle-densest subgraphs. In *2016 IEEE/ACM, ASONAM 2016, San Francisco, CA, USA, August 18-21, 2016*, pages 33–40, 2016.

[35] A. E. Sariyuce, C. Seshadhri, A. Pinar, and U. V. Catalyurek. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *WWW*, pages 927–937, 2015.

[36] J. G. Siek, L.-Q. Lee, and A. Lumsdaine. *Boost Graph Library: User Guide and Reference Manual, The.* Pearson Education, 2001.

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

[38] N. Tatti and A. Gionis. Density-friendly graph decomposition. In *WWW*, pages 1089–1099, 2015.

[39] C. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. Tsiarli. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *SIGKDD*, pages 104–112, 2013.

[40] B. W. Turnbull. The empirical distribution function with arbitrarily grouped, censored and truncated data. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 290–295, 1976.

[41] J. Wang and J. Cheng. Truss decomposition in massive networks. *Proceedings of the VLDB Endowment*, 5(9):812–823, 2012.

[42] 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.