Finding Dense Subgraphs with Size Bounds

Reid Andersen and Kumar Chellapilla

Microsoft Live Labs, Redmond WA 98052, USA
{reidan,kumarc}@microsoft.com

Abstract. We consider the problem of finding dense subgraphs with specified upper or lower bounds on the number of vertices. We introduce two optimization problems: the densest at-least-k-subgraph problem (`dalks`), which is to find an induced subgraph of highest average degree among all subgraphs with at least k vertices, and the densest at-most-k-subgraph problem (`damks`), which is defined similarly. These problems are relaxed versions of the well-known densest k-subgraph problem (`dks`), which is to find the densest subgraph with exactly k vertices. Our main result is that `dalks` can be approximated efficiently, even for web-scale graphs. We give a (1/3)-approximation algorithm for `dalks` that is based on the core decomposition of a graph, and that runs in time $O(m+n)$, where n is the number of nodes and m is the number of edges. In contrast, we show that `damks` is nearly as hard to approximate as the densest k-subgraph problem, for which no good approximation algorithm is known. In particular, we show that if there exists a polynomial time approximation algorithm for `damks` with approximation ratio γ, then there is a polynomial time approximation algorithm for `dks` with approximation ratio $\gamma^2/8$. In the experimental section, we test the algorithm for `dalks` on large publicly available web graphs. We observe that, in addition to producing near-optimal solutions for `dalks`, the algorithm also produces near-optimal solutions for `dks` for nearly all values of k.

1 Introduction

The density of an induced subgraph is the number of edges contained in the subgraph, divided by the number of vertices. Identifying subgraphs with high density is a useful primitive, which has been applied to find web communities, produce compressed representations of graphs, and identify link spam [9,14,20,8].

Effective heuristics have been developed to identify various kinds of dense subgraphs. Kumar et al. gave an algorithm for finding bipartite cliques [20]. Dourisboure et al. gave a scalable heuristic for finding small dense communities in web graphs [9]. The algorithm of Gibson et al. [14] finds dense communities using two-level min-hashing, with the goal of identifying link spam. Generally speaking, these algorithms are designed to find collections of small dense subgraphs that are isolated from each other, which are often viewed as the dense centers of communities in the graph. This is quite different from the task of finding a single large dense subgraph that contains a significant fraction of the graph, which is often used as a method of preprocessing or subsampling the graph.

K. Avrachenkov, D. Donato, and N. Litvak (Eds.): WAW 2009, LNCS 5427, pp. 25–37, 2009.

The complexity of identifying dense subgraphs can vary greatly when additional constraints on the size of the subgraph are introduced. Finding the densest subgraph with an arbitrary number of vertices is known as the densest subgraph problem `ds`, and can be solved exactly in polynomial time by solving a sequence of maximum flow problems [15,13]. The algorithm of Kortsarz and Peleg [18] produces a $(1/2)$-approximation of the densest subgraph in linear time, which is useful for graphs where the time required to compute maximum flows is prohibitively large. In contrast, no efficient algorithm is known for the problem of finding the densest subgraph with exactly k vertices, where k is specified as part of the input. This is the densest k-subgraph problem, or `dks`. The `dks` problem is $\mathcal{NP}$-complete, and the best polynomial time algorithm known for `dks` (due to Feige, Peleg, and Kortsarz) has the approximation ratio $n^{-(1/3)+\delta}$, for a small constant δ [11].

We want to control the size of the dense subgraphs we find, but we need to avoid the difficult task of finding dense subgraphs of a specified size. In this paper, we address this problem by introducing two variations of the densest subgraph problem: finding the densest subgraph with at least k vertices, and finding the densest subgraph with at most k vertices. We refer to these as the densest at-least-k-subgraph problem (`dalks`), and the densest at-most-k-subgraph problem (`damks`). These two relaxed versions of the densest k-subgraph problem roughly correspond to the two types of applications for dense subgraphs described earlier; for finding communities one would want an algorithm to solve `damks`, and for preprocessing a graph one would want an algorithm for `dalks`. Our main result is to show that `dalks` can be solved efficiently, while `damks` is nearly as hard to approximate as `dks`. In Section 3, we introduce a $(1/3)$-approximation algorithm for `dalks` that runs in time $O(m+n)$ in an unweighted graph. In Section 4, we prove a reduction that shows any polynomial time γ-approximation algorithm for `damks` can be used to design a polynomial time $(\gamma^2/8)$-approximation algorithm for `dks`.

Our algorithm for `dalks` is based on the *core decomposition*, and it can be viewed as a generalization of Kortsarz and Peleg's $(1/2)$-approximation algorithm for densest subgraph problem [18]. The core decomposition was first introduced as a tool for social network analysis [19]. It has been used in several applications, including graph drawing [2] and the analysis of biological networks [21].

In Section 6, we present experimental results for `dalks` on publicly available webgraphs. We demonstrate that the algorithm finds subgraphs with nearly optimal density while providing considerable control over the subgraph size. Surprisingly, we observe that on typical web graphs, the algorithm also produces a good approximation of the densest subgraph on exactly k vertices, for nearly all values of k. In Section 5, we describe theoretical results that help to explain this observation. We introduce two graph parameters based on the density of the graph's cores. Given these parameters, we prove bounds on the range of k for which our algorithm produces a good approximation for the densest k-subgraph problem.

1.1 Related Work

Here we survey results on the complexity of the densest k-subgraph problem. The best approximation algorithm known for the general problem (when k is specified as part of the input) is the algorithm of Feige, Peleg, and Kortsarz [11], which has approximation ratio $O(n^{-(1/3)+\delta})$, for a small constant $\delta > 0$. For any particular value of k, the greedy algorithm of Asahiro et al. [6] gives the ratio $O(k/n)$. Algorithms based on linear programming and semidefinite programming have produced approximation ratios better than $O(k/n)$ for certain values of k, but do not improve the approximation ratio for the general case [12,10].

Feige and Seltser [12] showed that `dks` is $\mathcal{NP}$-complete when restricted to bipartite graphs of maximum degree 3, by a reduction from max-clique. This reduction does not produce a hardness of approximation result for `dks`. In fact, they showed that if a graph contains a k-clique, then a subgraph with k vertices and $(1-\epsilon)\binom{k}{2}$ edges can be found in subexponential time. Khot [17] proved there can be no PTAS (polynomial time approximation scheme) for the densest k-subgraph problem, under a reasonable complexity assumption. Arora, Karger, and Karpinski [4] gave a PTAS for the special case $k = \Omega(n)$ and $m = \Omega(n^2)$. Asahiro, Hassin, and Iwama [5] showed that the problem is still $\mathcal{NP}$-complete for very sparse graphs.

Kannan and Vinay [16] introduced a different objective function for density, which is defined for a pair of vertex subsets S and T rather than a single subgraph. They gave an $O(\log n)$-approximation for this objective function using spectral techniques. Charikar [7] later gave a linear time $(1/2)$-approximation algorithm for this objective function, based on the core decomposition, and showed that the problem can be solved exactly by linear programming. A local algorithm for finding small subgraphs with high density according to the Kannan-Vinay objective function was described in [3].

2 Definitions

Let $G = (V, E)$ be an undirected graph with a weight function $w : E \to \mathbb{R}_+$ that assigns a positive weight to each edge. The weighted degree $w(v, G)$ is the sum of the weights of the edges in G incident with v. The total weight $W(G)$ is the sum of the weights of the edges in G.

Definition 1. *For any induced subgraph H of G, the* **density** $d(H)$ *of H is*

$$d(H) := \frac{W(H)}{|H|}.$$

Definition 2. *For an undirected graph G, we define the following quantities.*

$D_{al}(G, k) :=$ *the maximum density of any induced subgraph of G with at least $k$vertices.*

$D_{am}(G, k) :=$ *the maximum density of any induced subgraph of G with at most $k$vertices.*

$D_{eq}(G,k) :=$ *the maximum density of any induced subgraph of* G *with exactly* k*vertices.*

$D_{max}(G) :=$ *the maximum density of any induced subgraph of* G*.*

The densest at-least-k-subgraph problem (`dalks`) is to find an induced subgraph with at least k vertices achieving density $D_{al}(G,k)$. Similarly, the densest at-most-k-subgraph problem (`damks`) is to find an induced subgraph with at most k vertices achieving density $D_{am}(G,k)$. The densest k-subgraph problem (`dks`) is to find an induced subgraph with exactly k vertices achieving $D_{eq}(G,k)$, and the densest subgraph problem (`ds`) is to find an induced subgraph of any size achieving $D_{max}(G)$.

We now define what it means to be an approximation algorithm for `dalks`. Approximation algorithms for `damks`, `dks`, and `ds` are defined similarly.

Definition 3. *We say an algorithm* $A(G,k)$ *is a* γ-approximation algorithm *for the densest at-least-*k*-subgraph problem if, for any graph* G *and integer* k*, it returns an induced subgraph* $H \subseteq G$ *with at least* k *vertices and density* $d(H) \geq \gamma D_{al}(G,k)$*.*

3 Finding Dense Subgraphs with at Least k Vertices

In this section, we describe an algorithm `FindLargeDenseSubgraph` that is a (1/3)-approximation algorithm for the densest at-least-k-subgraph problem and that runs in time $O(m+n)$ in an unweighted graph. The algorithm is described in Table 1. The main step of the algorithm computes the *core decomposition* of the graph using a well-known greedy procedure (see [18,7,2]). This produces an ordering $(v_1,\ldots,v_n)$ of the vertices of the graph, after which the algorithm outputs a subgraphs of the form $\{v_1,\ldots,v_j\}$. Kortsarz and Peleg [18] used the core decomposition to give a (1/2)-approximation algorithm for `ds`. Theorem 1 extends their result to show that the core decomposition can be used to approximate `dalks`.

Theorem 1. `FindLargeDenseSubgraph`(G,k) *is a* (1/3)*-approximation algorithm for the densest at-least-*k*-subgraph problem.*

The proof of Theorem 1 is in Section 3.1.

The core decomposition procedure, which dominates the running time of `FindLargeDenseSubgraph`, can be implemented to run in time $O(m+n)$ in an unweighted graph and $O(m+n\log n)$ in a weighted graph. For a proof, we refer the reader to [18]. This implies the following proposition.

Proposition 1. *The running time of* `FindLargeDenseSubgraph`(G,k) *is* $O(m+n)$ *in an unweighted graph, and* $O(m+n\log n)$ *in a weighted graph.*

`FindLargeDenseSubgraph`(G, k) :
Input: a graph G with n vertices, and an integer k.
Output: an induced subgraph of G with at least k vertices.

1. Compute the core decomposition of G:
   Let $H_n = G$ and repeat the following for $i = n, \ldots, 1$,
   (a) Let r_i be the minimum weighted degree of any vertex in H_i.
   (b) Let v_i be a vertex of minimum weighted degree, where $w(v_i, H_i) = r_i$.
   (c) Remove v_i from H_i to form the induced subgraph H_{i-1}.
   (d) Update the values of $W(H_i)$ and $d(H_i)$ as follows,

$$W(H_{i-1}) = W(H_i) - 2r_i,$$

$$d(H_{i-1}) = W(H_{i-1})/(i-1).$$

   Note that part 1 produces an ordering of the vertices $v_1, \ldots, v_n$, where v_1 is the last vertex removed and v_n is the first. The set H_i consists of the vertices $\{v_1, \ldots, v_i\}$.
2. Output the subgraph H_i with the largest density $d(H_i)$ over all $i \geq k$.

Fig. 1. Description of `FindLargeDenseSubgraph`

3.1 Analysis of the Algorithm

To analyze `FindLargeDenseSubgraph`, we consider the relationship between induced subgraphs of G with high average degree (dense subgraphs) and induced subgraphs of G with high minimum degree (w-cores).

Definition 4. *Given a graph G and a weight $w \in \mathbb{R}$, the* **w-core** *$C_w(G)$ is the unique largest induced subgraph of G with minimum weighted degree at least w.*

Here is an outline of how we will proceed. We first show that the `FindLargeDenseSubgraph` algorithm computes all the w-cores of G (Lemma 1). We then show that for any induced subgraph H of G with density d, the $(2d/3)$-core of G has total weight at least $W(H)/3$ (Lemma 2). We prove Theorem 1 using these two lemmas.

Lemma 1. *Let $\{H_1, \ldots, H_n\}$, and $\{r_1, \ldots, r_n\}$ be the induced subgraphs and weighted degrees determined by the algorithm* `FindLargeDenseSubgraph` *on the input graph G. For any $w \in \mathbb{R}$, let $I(w)$ be the largest index such that $r_{I(w)} \geq w$. Then, $H_{I(w)} = C_w(G)$. In other words, every w-core of G is equal to one of the subgraphs H_i.*

Proof. Fix a value of w. It easy to see (by induction) that none of the vertices $v_n \ldots v_{I(w)+1}$ that were removed before $v_{I(w)}$ can be contained in an induced subgraph with minimum degree at least w. That implies $C_w(G) \subseteq H_{I(w)}$. On the other hand, the minimum degree of $H_{I(w)}$ is at least w, so $H_{I(w)} \subseteq C_w(G)$. Therefore, $H_{I(w)} = C_w(G)$. □

Lemma 2. *For any graph G with n nodes, total weight W, and density $d = W/n$, the d-core of G is nonempty. Furthermore, for any $\alpha \in [0,1]$, the total weight of the (αd)-core of G is strictly greater than $(1-\alpha)W$.*

Proof. Let $\{H_1, \ldots, H_n\}$ be the induced subgraphs determined by `FindLargeDenseSubgraph` on the input graph G. Fix a value of w, and let $I(w)$ be the largest index such that $r_{I(w)} \geq w$. Recall that $H_{I(w)} = C_w(G)$ by Lemma 1. Since each edge in G is removed once during the course of the algorithm,

$$\begin{aligned} W &= \sum_{i=1}^{n} r_i \\ &= \sum_{i=1}^{I(w)} r_i + \sum_{i=I(w)+1}^{n} r_i \\ &< W(H_{I(w)}) + w \cdot (n - I(w)) \\ &\leq W(C_w(G)) + w \cdot n. \end{aligned}$$

Therefore,

$$W(C_w(G)) > W - w \cdot n.$$

Taking $w = d = W/n$ in the equation above, we learn that $W(C_d(G)) > 0$. Taking $w = \alpha d = \alpha W/n$, we learn that $W(C_{\alpha d}(G)) > (1-\alpha)W$. □

Proof (Theorem 1). Let $\{H_1, \ldots, H_n\}$ be the induced subgraphs computed by `FindLargeDenseSubgraph` on the input graph G. It suffices to show that for any k, there is an integer $I \in [k, n]$ satisfying $d(H_I) \geq D_{al}(G,k)/3$.

Let H_* be an induced subgraph of G with at least k vertices and with density $d_* = W(H_*)/|H_*| = D_{al}(G,k)$. We apply Lemma 2 to H_* with $\alpha = 2/3$ to show that $C_{(2d_*/3)}(H_*)$ has total weight at least $W(H_*)/3$. This implies that $C_{(2d_*/3)}(G)$ has total weight at least $W(H_*)/3$.

The core $C_{(2d_*/3)}(G)$ has minimum degree at least $2d_*/3$, so its density is at least $d_*/3$. Lemma 1 shows $C_{(2d_*/3)}(G) = H_I$, for $I = |C_{(2d_*/3)}(G)|$. If $I \geq k$, then H_I satisfies the requirements of the theorem. If $I < k$, then $C_{(2d_*/3)}(G) = H_I$ is contained in H_k, and the following calculation shows that H_k satisfies the requirements of the theorem.

$$d(H_k) = \frac{W(H_k)}{k} \geq \frac{W(C_{(2d_*/3)}(G))}{k} \geq \frac{W(H_*)/3}{k} = d_*/3.$$ □

We remark that our analysis of `FindLargeDenseSubgraph` is a generalization of the result of Kortsarz-Peleg [18]. Their result shows that `FindLargeDenseSubgraph`$(G,1)$ is a $(1/2)$-approximation algorithm for `ds`. This follows from the fact that if $w = D_{max}(G)$, then the w-core of G is nonempty, which is a special case of Lemma 2.

4 Finding Dense Subgraphs with at Most k Vertices

The densest at-most-k-subgraph problem is $\mathcal{NP}$-complete by a reduction to the max-clique problem, since a subgraph of size at most k has density at least $(k-1)/2$ if and only if it is a k-clique. Feige and Seltser [12] proved that the densest k-subgraph problem is $\mathcal{NP}$-complete even when restricted to graphs with maximum degree 3, and their proof implies that the densest at-most-k-subgraph problem is $\mathcal{NP}$-complete when restricted to the same class of graphs.

In this section, we show that `damks` is nearly as hard to approximate as `dks`. We show that if there exists a polynomial time *pseudo-approximation* algorithm for `damks`, which outputs a set of at most βk vertices with density at least γ times the density of the densest subgraph with at most k vertices, then there exists a polynomial time approximation algorithm for `dks` with ratio $\gamma \min(\gamma, \beta^{-1})/8$. As an immediate consequence, a polynomial time γ-approximation algorithm for `damks` would imply a polynomial time $(\gamma^2/8)$-approximation algorithm for `dks`.

Definition 5. *An algorithm $A(G,k)$ is a (β,γ)-*algorithm *for the densest at-most-k-subgraph problem if for any input graph G and integer k, it returns an induced subgraph of G with at most βk vertices and density at least $\gamma D_{am}(G,k)$.*

Theorem 2. *If there is a polynomial time (β,γ)-algorithm for the densest at-most-k-subgraph problem (where $\beta \geq 1$ and $\gamma \leq 1$), then there is a polynomial time $(\gamma \min(\gamma, \beta^{-1})/8)$-approximation algorithm for the densest k-subgraph problem.*

Proof. Assume there exists a polynomial time algorithm $A(G,k)$ that is (β,γ)-algorithm for `damks`. We will now describe a polynomial time approximation algorithm for `dks`.

Given as input a graph G and integer k, let $G_1 = G$, let $i = 1$, and repeat the following procedure. Let $H_i = A(G_i, k)$ be an induced subgraph of G_i with at most βk vertices and with density at least $\gamma D_{am}(G_i, k)$. Remove all the edges in H_i from G_i to form a new graph G_{i+1} on the same vertex set as G. Repeat this procedure until all edges have been removed from G.

Let n_i be the number of vertices in H_i, let $W_i = W(H_i)$, and let $d_i = d(H_i) = W_i/n_i$. Let H_* be an induced subgraph of G with exactly k vertices and density $d_* = D_{eq}(G,k)$. Notice that if $(W_1 + \cdots + W_{t-1}) \leq W(H_*)/2$, then $d_t \geq \gamma d_*/2$. This is true because d_t is at least γ times the density of the induced subgraph of G_t on the vertex set of H_*, which is at least

$$\frac{W(H_*) - (W_1 + \cdots + W_{t-1})}{k} \geq \frac{W(H_*)}{2k} = \frac{d_*}{2}.$$

Let T be the smallest integer such that $(W_1 + \cdots + W_T) \geq W(H_*)/2$, and let U_T be the induced subgraph on the union of the vertex sets of $H_1, \ldots, H_T$. The total weight $W(U_T)$ is at least $W(H_*)/2$. The density of U_T is

$$d(U_T) = \frac{W(U_T)}{|U_T|} \geq \frac{W_1 + \cdots + W_T}{n_1 + \cdots + n_T} \geq \min_{1 \leq t \leq T} \frac{W_t}{n_t} \geq \gamma \frac{d_*}{2}.$$

To bound the number of vertices in U_T, notice that $(n_1 + \cdots + n_{T-1}) \leq \gamma^{-1}k$, because

$$\frac{d_* k}{2} = \frac{W(H_*)}{2} \geq \sum_{i=1}^{T-1} W_i = \sum_{i=1}^{T-1} n_i d_i \geq \gamma \frac{d_*}{2} \sum_{i=1}^{T-1} n_i.$$

Since n_T is at most βk, we have $|U_T| \leq (n_1 + \cdots + n_T) \leq (\gamma^{-1} + \beta)k$.

There are now two cases to consider. If $|U_T| \leq k$, then we pad U_T with arbitrary vertices to form a set U'_T of size exactly k. The set U'_T is still sufficiently dense:

$$d(U'_T) \geq \frac{W(H_*)/2}{k} = \frac{d_*}{2}.$$

If $|U_T| > k$, then we employ a simple greedy procedure to reduce the number of vertices. We begin with the induced subgraph U_T, greedily remove the vertex with smallest degree to obtain a smaller subgraph, and repeat until exactly k vertices remain. The resulting subgraph U''_T has density at least $d(U_T)(k/2|U_T|)$ by the method of conditional expectations (this technique was also used in [11]). The set U''_T is sufficiently dense:

$$\begin{aligned} d(U''_T) \geq d(U_T)\frac{k}{2|U_T|} &\geq \gamma \frac{d_*}{2}\left(\frac{k}{2(\gamma^{-1}+\beta)k}\right) = d_* \frac{\gamma}{4(\gamma^{-1}+\beta)} \\ &\geq d_* \frac{\gamma}{8\max(\gamma^{-1},\beta)} = d_* \frac{\gamma \min(\gamma, \beta^{-1})}{8}. \end{aligned}$$

□

5 Finding Dense Subgraphs of Specified Size

The previous section shows that the densest at-least-k subgraph problem is easy to approximate within a constant factor for any graph and any value of k. The densest k-subgraph problem seems hard to approximate well in the worst case, but we may still be able to find near-optimal solutions for specific instances. In this section we describe a method for identifying a range of k-values for which we can obtain a good approximation of the densest k-subgraph.

Here is an outline of our approach. We first define a graph parameter $k_*(G) \in [1, n]$. We then prove that for all $k \geq k_*$, the algorithm `FindLargeDenseSubgraph` can be used to find a (1/3)-approximation of the densest subgraph with exactly k vertices. In Section 6, we observe empirically that for several example web graphs, the value of k_* is only a small fraction of n.

Definition 6. *For a given graph G, let w_* be the smallest value such that the average degree of the core $C(w_*)$ is less than $2w_*$. Let $k_*(G) = |C(w_*)|$ be the number of vertices in that core.*

Roughly speaking, k_* describes how small a core of the graph must be before it can be nearly degree-regular. The following theorem shows that for every $k \geq k_*$, the set H_k produced by `FindLargeDenseSubgraph` has density at least 1/3 of the densest k-subgraph.

Theorem 3. *Let* $\{v_1, \ldots, v_n\}$ *be the ordering of the vertices produced by* `FindLargeDenseSubgraph`*, and let* $H_k = \{v_1, \ldots, v_k\}$*. Then, for any* $k \geq k_*$ *we have* $d(H_k) \geq (1/3)D_{eq}(G, k)$*.*

Proof. We will first show the following:

$$d(H_{k+1}) \leq d(H_k) \text{ for all } k \geq k_*. \tag{1}$$

Once we show this, then for any $k \geq k_*$ we have

$$d(H_k) = \max_{j \geq k} H_j \geq \frac{1}{3} D_{al}(k) \geq \frac{1}{3} D_{eq}(k).$$

The middle step follows from the approximation guarantee proved in Theorem 1.

To prove (1), it suffices to take an arbitrary value of w for which $|C_w| > k_*$, and show that $d(H_{j-1}) \geq d(H_j)$ for all j in the interval $(|C_{w+1}|, |C_w|]$. We prove this by induction, first assuming $d(H_j) \geq d(C_w)$ and then proving $d(H_{j-1}) \geq d(C_w)$. Recall that $r(j)$ is the degree of v_j in H_j. Then,

$$d(H_{j-1}) = \frac{j \cdot d(H_j) - 2r(j)}{j-1} \geq \frac{j \cdot d(H_j) - d(H_j)}{j-1} = d(H_j).$$

Here we used the fact that $2r(j) \leq d(H_j)$. This is true because $r(j) \leq w$, our assumption that $|C_w| \geq k_*$ implies $w \leq d(C_w)/2$, and our induction assumption implies $d(C_w)/2 \leq d(H_j)/2$. □

When $k < k_*$ the previous theorem doesn't apply, but we can still bound the ratio between $d(H_k)$ and the optimal density $D_{eq}(k)$. The following bound holds for any $k \in [1, n]$, and can be computed easily by observing the densities of the sets $H_1, \ldots, H_n$.

Lemma 3. *Let* $R_k = \max_{j \geq k} \frac{d(H_j)}{d(H_k)}$*. For any value of* k*, we have* $d(H_k) \geq \frac{R_k}{3} D_{eq}(G, k)$*.*

Proof. For any value of k, we have

$$d(H_k) = R_k \max_{j \geq k} d(H_j) \geq \frac{R_k}{3} D_{al}(k) \geq \frac{R_k}{3} D_{eq}(k).$$

The middle step follows from the approximation guarantee proved in Theorem 1. □

We remark that to prove Theorem 3, we showed that $R_k = 1$ for all $k \geq k_*$. In the next section we will compute the values of k_* and R_k for several example graphs.

6 Experiments

In this section we present experimental results on four example graphs. The graphs and their sizes are listed in Table 1. Three of these graphs are publicly

Table 1. Graph size, running time, and the observed value of k_*

graph	num nodes (n)	total degree (2m)	running time (sec)	k_*
domain-2006	55,554,153	1,067,392,106	263.81	9,445
webbase-2001	118,142,156	1,985,689,782	204.573	48,190
uk-2005	39,459,926	1,842,690,156	92.271	368,741
cnr-2000	325,558	6,257,420	0.359	13,237

Table 2. Attributes of the densest core, highest core, and w_* core in the four test graphs

graph	core number	nodes in core	density	worst R_k
	w	$\|C_w\|$	$d(C_w)$	$\max_{k \geq \|C_w\|} R_k$
domain-2006				
w_* core	1099	9445	2196.32	1
densest core	1203	4737	2275.96	.9694
highest core	1298	2502	2072.42	.9104
webbase-2001				
w_* core	548	48190	1089.42	1
densest core	2281	1219	2436	.8547
highest core	2281	1219	2436	.8547
uk-2005				
w_* core	258	368741	515.851	1
densest core	1002	587	1171.98	.8871
highest core	1002	587	1171.98	.8871
cnr-2000				
w_* core	38	13237	75.1145	1
densest core	116	82	161.976	.9138
highest core	116	82	161.976	.9138

available webgraphs from the *Laboratory for Web Algorithmics*[1] at the *Univerita Degli Studi Di Milano*. The graph *webbase-2001* was obtained from the 2001 crawl performed by the WebBase crawler. The graph *uk-2005* was obtained from a 2005 crawl of the .uk domain, performed by UbiCrawler. The graph *cnr-2000* was obtained from a small crawl of the Italian CNR domain. These graphs were chosen because they are fairly large, easy to obtain, and have been used in previous research papers. The remaining graph *domain-2006* is a snapshot of the domain graph in September 2006, from Microsoft.

These were originally directed graphs, but we have treated them as undirected graphs in the following way. We consider a directed link from a vertex u to a vertex v as an undirected link between u and v. We remark that there will be a link with multiplicity 2 between u and v in this undirected graph if both (u, v) and (v, u) appeared in the original directed graph. For this reason, the average degree of a subgraph on k vertices may be as large as $2(k-1)$. In addition, we

[1] http://law.dsi.unimi.it/

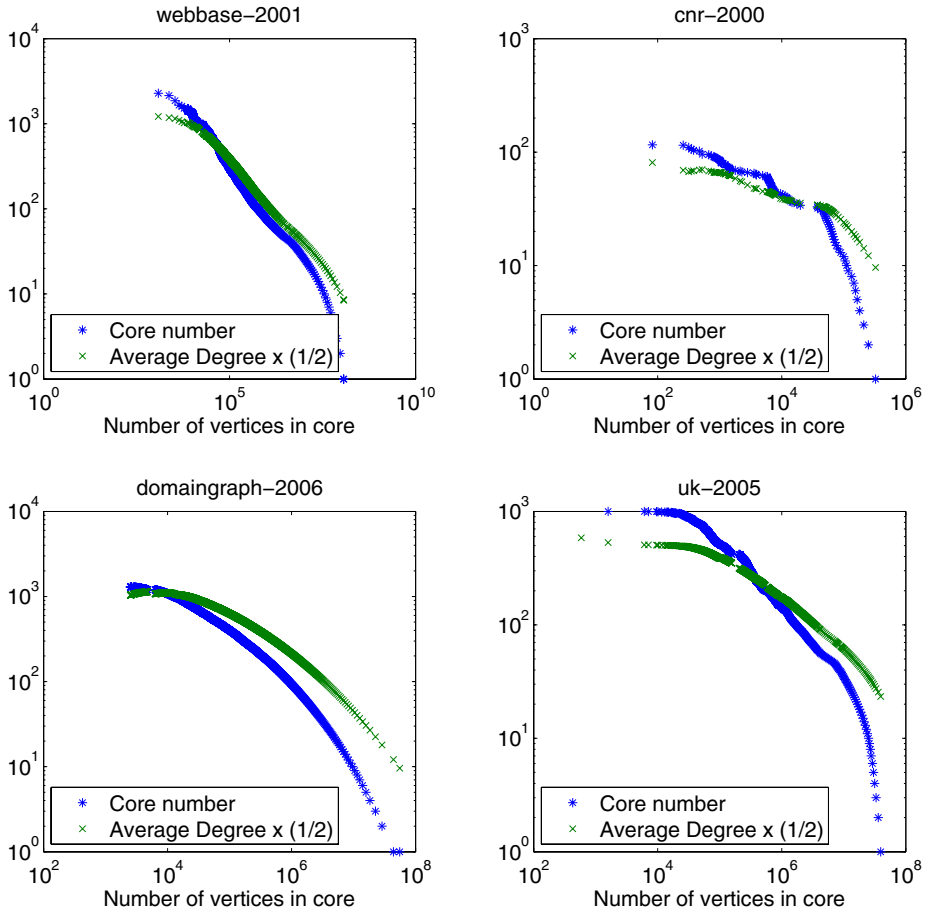

Fig. 2. Plots of core size versus core number and core density in four webgraphs

removed all self-loops from the graphs. The total degrees reported in Table 1 were computed after these modifications were made.

In Table 1, we report running times for our implementation of `FindLargeDenseSubgraph`. We implemented the algorithm in C++, and ran our experiments on a single machine with 64GB of RAM and a 3.0Ghz quad-core Intel Xeon processor. Only one of the processor cores was used by the algorithm. The time we report is the time required to compute the core decomposition, which produces an ordering of all vertices in the graph. The running time does not include the time required to load the graph from disk into memory.

We also report in Table 1 the values of k_* for each of these graphs. We observe that k_* is small compared to the number of vertices in the graph, which is good because our algorithm produces a good approximation of the densest k-subgraph for all k larger than k_*. In Table 2 we report statistics for three special cores in each of the example graphs. We report the w_*-core (see Definition 6), which is

the core that determines the value of k_*. We report the core that has the highest density (densest core), and we also report the highest value of w for which the w-core is nonempty (highest core). Note that these last two are not the same in general, but they end up being the same for three of our example graphs. For each of these special cores we report the w-value of the core, the number of vertices in the core, and the density of the core.

For each of the cores in Table 2 we also report a statistic regarding the quantity R_k described in Lemma 3. For each core C_w we report "worst R_k", which we define to be the smallest value of R_k over all values of $k \geq |C_w|$. The table indicates that "worst R_k" is close to 1 for the highest core, which means we can approximate `dks` well for all values of k above the size of the highest core. For example, in the graph domain-2006, the highest core contains 2502 nodes and has a value of .9104 for worst R_k. That means for all values of $k \geq 2502$, the set H_k produced by `FindLargeDenseSubgraph` on domain-2006 is within a factor of $.9104 * 1/3$ of the densest subgraph on exactly k vertices, by Lemma 3.

Figure 2 contains a plot for each of the four webgraphs that shows the size, core number, and density of all of the graph's cores. Each of the plots in the figure has two curves. Each point on the curve represents a w-core. One curve shows the core number, the other shows the density of the core, and both are plotted against the number of vertices in the core. The value of k_* can be seen from these plots; it is the x-coordinate of the first point (from right to left) at which these two curves intersect.

References

1. Abello, J., Resende, M.G.C., Sudarsky, R.: Massive quasi-clique detection. In: Rajsbaum, S. (ed.) LATIN 2002. LNCS, vol. 2286, pp. 598–612. Springer, Heidelberg (2002)
2. Alvarez-Hamelin, J.I., Dall'Asta, L., Barrat, A., Vespignani, A.: Large scale networks fingerprinting and visualization using the k-core decomposition. Advances in Neural Information Processing Systems 18, 41–50 (2006)
3. Andersen, R.: A local algorithm for finding dense subgraphs. In: Proc. 19th ACM-SIAM Symposium on Discrete Algorithms (SODA 2008), pp. 1003–1009 (2008)
4. Arora, S., Karger, D., Karpinski, M.: Polynomial time approximation schemes for dense instances of NP-hard problems. In: Proc. 27th ACM Symposium on Theory of Computing (STOC 1995), pp. 284–293 (1995)
5. Asahiro, Y., Hassin, R., Iwama, K.: Complexity of finding dense subgraphs. Discrete Appl. Math. 121(1-3), 15–26 (2002)
6. Asahiro, Y., Iwama, K., Tamaki, H., Tokuyama, T.: Greedily finding a dense subgraph. J. Algorithms 34(2), 203–221 (2000)
7. Charikar, M.: Greedy approximation algorithms for finding dense components in a graph. In: Jansen, K., Khuller, S. (eds.) APPROX 2000. LNCS, vol. 1913, pp. 84–95. Springer, Heidelberg (2000)
8. Buehrer, G., Chellapilla, K.: A scalable pattern mining approach to web graph compression with communities. In: WSDM 2008: Proceedings of the international conference on web search and web data mining, pp. 95–106 (2008)
9. Dourisboure, Y., Geraci, F., Pellegrini, M.: Extraction and classification of dense communities in the web. In: WWW 2007: Proceedings of the 16th international conference on World Wide Web, pp. 461–470 (2007)

10. Feige, U., Langberg, M.: Approximation algorithms for maximization problems arising in graph partitioning. J. Algorithms 41(2), 174–211 (2001)
11. Feige, U., Peleg, D., Kortsarz, G.: The dense k-subgraph problem. Algorithmica 29(3), 410–421 (2001)
12. Feige, U., Seltser, M.: On the densest k-subgraph problem, Technical report, Department of Applied Mathematics and Computer Science, The Weizmann Institute, Rehobot (1997)
13. Gallo, G., Grigoriadis, M., Tarjan, R.: A fast parametric maximum flow algorithm and applications. SIAM J. Comput. 18(1), 30–55 (1989)
14. Gibson, D., Kumar, R., Tomkins, A.: Discovering large dense subgraphs in massive graphs. In: Proc. 31st VLDB Conference (2005)
15. Goldberg, A.: Finding a maximum density subgraph, Technical Report UCB/CSB 84/171, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA (1984)
16. Kannan, R., Vinay, V.: Analyzing the structure of large graphs (manuscript) (1999)
17. Khot, S.: Ruling out PTAS for graph min-bisection, dense k-subgraph, and bipartite clique. SIAM Journal on Computing 36(4), 1025–1071 (2006)
18. Kortsarz, G., Peleg, D.: Generating sparse 2-spanners. J. Algorithms 17(2), 222–236 (1994)
19. Seidman, S.B.: Network structure and minimum degree. Social Networks 5, 269–287 (1983)
20. Kumar, R., Raghavan, P., Rajagopalan, S., Tomkins, A.: Trawling the Web for emerging cyber-communities. In: Proc. 8th WWW Conference (WWW 1999) (1999)
21. Wuchty, S., Almaas, E.: Peeling the yeast protein network. Proteomics 5, 444 (2005)