Greedy Approximation Algorithms for Finding Dense Components in a Graph

Moses Charikar*

Stanford University, Stanford, CA 94305, USA.
moses@cs.stanford.edu

Abstract. We study the problem of finding highly connected subgraphs of undirected and directed graphs. For undirected graphs, the notion of density of a subgraph we use is the average degree of the subgraph. For directed graphs, a corresponding notion of density was introduced recently by Kannan and Vinay. This is designed to quantify highly connectedness of substructures in a sparse directed graph such as the web graph. We study the optimization problems of finding subgraphs maximizing these notions of density for undirected and directed graphs. This paper gives simple greedy approximation algorithms for these optimization problems. We also answer an open question about the complexity of the optimization problem for directed graphs.

1 Introduction

The problem of finding dense components in a graph has been extensively studied [1, 2, 4, 5, 9]. Researchers have explored different definitions of density and examined the optimization problems corresponding to finding substructures that maximize a given notion of density. The complexity of such optimization problems varies widely with the specific choice of a definition. In this paper, the notion of density we will be interested in is, loosely speaking, the average degree of a subgraph. Precise definitions for both undirected and directed graphs appear in Section 1.1.

Recently, the problem of finding relatively highly connected sub-structures in the web graph has received a lot of attention [8, 10–12]. Experiments suggest that such substructures correspond to communities on the web, i.e. collections of pages related to the same topic. Further, the presence of a large density of links within a particular set of pages is considered an indication of the importance of these pages. The algorithm of Kleinberg [10] identifies *hubs* (resource lists) and *authorities* (authoritative pages) amongst the set of potential pages relevant to a query. The *hubs* are characterized by the presence of a large number of links to the *authorities* and the *authorities* are characterized by the presence of a large number of links from the *hubs*.

* Research supported by the Pierre and Christine Lamond Fellowship, an ARO MURI Grant DAAH04–96–1–0007 and NSF Grant IIS-9811904. Part of this work was done while the author was visiting IBM Almaden Research Center.

Kannan and Vinay [9] introduce a notion of density for directed graphs that quantifies relatively highly connected and is suitable for sparse directed graphs such as the web graph. This is motivated by trying to formalize the notion of finding sets of hubs and authorities that are highly connected relative to the rest of the graph. In this paper, we study the optimization problem of finding a subgraph of maximum density according to this notion. We now proceed to formally define the notions of density that we will use in this paper. These are identical to the definitions in [9].

1.1 Definitions and Notation

Let $G(V, E)$ be an undirected graph and $S \subseteq V$. We define $E(S)$ to be the edges induced by S, i.e.

$$E(S) = \{ij \in E : i \in S, j \in S\}$$

Definition 1. *Let $S \subseteq V$. We define the density $f(S)$ of the subset S to be*

$$f(S) = \frac{|E(S)|}{|S|}$$

We define the density $f(G)$ of the undirected graph $G(V, E)$ to be

$$f(G) = \max_{S \subseteq V}\{f(S)\}$$

Note that $2f(S)$ is simply the average degree of the subgraph induced by S and $2f(G)$ is the maximum average degree over all induced subgraphs. The problem of computing $f(G)$ is also known as the *Densest Subgraph* problem and can be solved using flow techniques. (See Chapter 4 in Lawler's book [13]. The algorithm, due to Gallo, Grigoriadis and Tarjan [7] uses parametric maximum flow which can be done in the time required to do a single maximum flow computation using the push-relabel algorithm).

A related problem that was been extensively studied is the *Densest k-Subgraph Problem*, where the goal is to find an induced subgraph of k vertices of maximum average degree [1, 2, 4, 5]. Relatively little is known about the approximability of this problem and resolving this remains a very interesting open question.

We now define density for directed graphs. Let $G(V, E)$ be a directed graph and $S, T \subseteq V$. We define $E(S, T)$ to be the set of edges going from S to T, i.e.

$$E(S, T) = \{ij \in E : i \in S, j \in T\}.$$

Definition 2. *Let $S, T \subseteq V$. We define the density $d(S, T)$ of the pair of sets S, T to be*

$$d(S, T) = \frac{|E(S, T)|}{\sqrt{|S||T|}}$$

We define the density $d(G)$ of the directed graph $G(V, E)$ to be

$$d(G) = \max_{S, T \subseteq V}\{d(S, T)\}$$

The above notion of density for directed graphs was introduced by Kannan and Vinay [9]. The set S corresponds to the hubs and the set T corresponds to the authorities in [10]. Note that for $|S| = |T|$, $d(S, T)$ is simply the average number of edges going from a vertex in S to T (or the average number of edges going into a vertex in T from S). Kannan and Vinay explain why this definition of density makes sense in the context of sparse directed graphs such as the web graph. Note that in the above definition, the sets S and T are not required to be disjoint.

The problem of computing $d(G)$ was considered in [9]. They obtain an $O(\log n)$ approximation by relating $d(G)$ to the singular value of the adjacency matrix of G and using the recently developed Monte Carlo algorithm for the Singular Value Decomposition of a matrix [3, 6]. They also show how the SVD techniques can be used to get an $O(\log n)$ approximation for $f(G)$. They leave open the question of resolving the complexity of computing $d(G)$ exactly.

In this paper, we prove that the quantity $d(G)$ can be computed exactly using linear programming techniques. We also give a simple greedy 2-approximation algorithm for this problem. As a warmup, we first explain how $f(G)$ can be computed exactly using linear programming techniques. We then present a simple greedy 2-approximation algorithm for this problem. This proceeds by repeatedly deleting the lowest degree vertex. Our algorithm and analysis for computing the density $d(G)$ for directed graphs builds on the techniques for computing $f(G)$ for undirected graphs.

2 Exact Algorithm for $f(G)$

We show that the problem of computing $f(G)$ can be expressed as a linear program. We will show that the optimal solution to this LP is a convex combination of integral solutions. This result by itself is probably not that interesting given the flow based exact algorithm for computing $f(G)$[7]. However, the proof technique will lay the foundation for the more complicated proofs in the algorithm for computing $d(G)$ later.

We use the following LP:

$$\max \sum_{ij} x_{ij} \tag{1}$$

$$\forall ij \in E \qquad x_{ij} \le y_i \tag{2}$$

$$\forall ij \in E \qquad x_{ij} \le y_j \tag{3}$$

$$\sum_i y_i \le 1 \tag{4}$$

$$x_{ij}, y_i \ge 0 \tag{5}$$

Lemma 1. *For any $S \subseteq V$, the value of the LP (1)-(5) is at least $f(S)$.*

Proof. We will give a feasible solution for the LP with value $f(S)$. Let $x = \frac{1}{|S|}$. For each $i \in S$, set $\bar{y}_i = x$. For each $ij \in E(S)$, set $\bar{x}_{ij} = x$. All the remaining variables are set to 0. Now, $\sum_i \bar{y}_i = |S| \cdot x = 1$. Thus, $(\bar{x}, \bar{y})$ is a feasible solution to the LP. The value of this solution is

$$|E(S)| \cdot x = \frac{|E(S)|}{|S|} = f(S)$$

This proves the lemma.

Lemma 2. *Given a feasible solution of the LP (1)-(5) with value v we can construct $S \subseteq V$ such that $f(S) \geq v$.*

Proof. Consider a feasible solution $(\bar{x}, \bar{y})$ to the LP (1)-(5). Without loss of generality, we can assume that for all ij, $\bar{x}_{ij} = \min(\bar{y}_i, \bar{y}_j)$.

We define a collection of sets S indexed by a parameter $r \geq 0$. Let $S(r) = \{i : \bar{y}_i \geq r\}$ and $E(r) = \{ij : \bar{x}_{ij} \geq r\}$. Since $\bar{x}_{ij} \leq \bar{y}_i$ and $\bar{x}_{ij} \leq \bar{y}_j$, $ij \in E(r) \Rightarrow i \in S(r), j \in S(r)$. Also, since $\bar{x}_{ij} = \min(\bar{y}_i, \bar{y}_j)$, $i \in S(r), j \in S(r) \Rightarrow ij \in E(r)$. Thus $E(r)$ is precisely the set of edges induced by $S(r)$.

Now, $\int_0^\infty |S(r)| dr = \sum_i \bar{y}_i \leq 1$. Note that $\int_0^\infty |E(r)| dr = \sum_{ij} \bar{x}_{ij}$. This is the objective function value of the LP solution. Let this value be v.

We claim that there exists r such that $|E(r)|/|S(r)| \geq v$. Suppose there were no such r. Then

$$\int_0^\infty |E(r)| dr < v \int_0^\infty |S(r)| dr \leq v.$$

This gives a contradiction. To find such an r, notice that we can check all combinatorially distinct sets $S(r)$ by simply checking the sets $S(r)$ obtained by setting $r = \bar{y}_i$ for every $i \in V$.

Putting Lemmas 1 and 2 together, we get the following theorem.

Theorem 1.

$$\max_{S \subseteq V} \{f(S)\} = OPT(LP) \tag{6}$$

where $OPT(LP)$ denotes the value of the optimal solution to the LP (1)-(5). Further, a set S maximizing $f(S)$ can be computed from the optimal solution to the LP.

Proof. First we establish the equality (6). From Lemma 1, the RHS $\geq$ the LHS. (Consider the S that maximizes $f(S)$). From Lemma 2, the LHS $\geq$ the RHS. The proof of Lemma 6 gives a construction of a set S that maximizes $f(S)$ from the optimal LP solution.

3 Greedy 2-approximation for $f(G)$

We want to produce a subgraph of G of large average degree. Intuitively, we should throw away low degree vertices in order to produce such a subgraph. This suggests a fairly natural greedy algorithm. In fact, the performance of such an algorithm has been analyzed by Asahiro, Iwama, Tamaki and Tokuyama [2] for a slightly different problem, that of obtaining a large average degree subgraph on a given number k of vertices.

The algorithm maintains a subset S of vertices. Initially $S \leftarrow V$. In each iteration, the algorithm identifies $i_{\min}$, the vertex of minimum degree in the subgraph induced by S. The algorithm removes $i_{\min}$ from the set S and moves on to the next iteration. The algorithm stops when the set S is empty. Of all the sets S constructed during the execution of the algorithm, the set S maximizing $f(S)$ (i.e. the set of maximum average degree) is returned as the output of the algorithm.

We will prove that the algorithm produces a 2 approximation for $f(G)$. There are various ways of proving this. We present a proof which may seem complicated at first. This will set the stage for the algorithm for $d(G)$ later. Moreover, we believe the proof is interesting because it makes connections between the greedy algorithm and the dual of the LP formulation we used in the previous section.

In order to analyze the algorithm, we produce an upper bound on the optimal solution. The upper bound has the following form: We assign each edge ij to either i or j. For a vertex i, $d(i)$ is the number of edges ij or ji assigned to i. Let $d^{\max} = \max_i\{d(i)\}$. (Another way to view this is that we will orient the edges of the graph and $d^{\max}$ is the maximum number of edges oriented towards any vertex). The following lemma shows that $f(S)$ is bounded by $d^{\max}$.

Lemma 3.

$$\max_{S \subseteq V}\{f(S)\} \leq d^{\max}$$

Proof. Consider the set S that maximizes $f(S)$. Now, each edge in $E(S)$ must be assigned to a vertex in S. Thus

$$|E(S)| \leq |S| \cdot d^{\max}$$
$$f(S) = \frac{|E(S)|}{|S|} \leq d^{\max}$$

This concludes the proof.

Now, the assignment of edges to one of the end points is constructed as the algorithm executes. Initially, all edges are unassigned. When the minimum degree vertex is deleted from S, the vertex is assigned all edges that go from the vertex to the rest of the vertices in S. We maintain the invariant that all edges between two vertices in the current set S are unassigned; all other edges are assigned. At the end of the execution of the algorithm, all edges are assigned.

Let $d^{\max}$ be defined as before for the specific assignment constructed corresponding to the execution of the greedy algorithm. The following lemma relates the value of the solution constructed by the greedy algorithm to $d^{\max}$.

Lemma 4. *Let v be the maximum value of $f(S)$ for all sets S obtained during the execution of the greedy algorithm. Then $d^{\max} \leq 2v$.*

Proof. Consider a single iteration of the greedy algorithm. Since $i_{\min}$ is selected to be the minimum degree vertex in S, its degree is at most $2|E(S)|/|S| \leq 2v$. Note that a particular vertex gets assigned edges to it only at the point when it is removed from S. This proves that $d^{\max} \leq 2v$.

Putting Lemmas 3 and 4 together, we get the following.

Theorem 2. *The greedy algorithm gives a* 2 *approximation for* $f(G)$*.*

Running Time It is easy to see that the greedy algorithm can be implemented to run in $O(n^2)$ time for a graph with n vertices and m edges. We can maintain the degrees of the vertices in the subgraph induced by S. Each iteration involves identifying and removing the minimum degree vertex as well as updating the degrees of the remaining vertices both of which can be done in $O(n)$ time. In fact, we can implement the algorithm to run in linear time. Since the degree of a vertex is an integer $\in \{0, n\}$, we can maintain lists of vertices with the same degree, i.e. a list of vertices of degree $0, 1, 2$ and so on. In each iteration, the minimum degree vertex is removed and degrees of the neighboring vertices updated, requiring them to be moved to new lists. The total work done in these updates is $O(m)$. Note that the minimum degree drops by at most 1 in each iteration. If the minimum degree in a particular iteration was d, the minimum degree vertex for the next iteration is obtained by scanning the lists for degrees $d-1$, d, $d+1$ and so on. The total work done in scanning these lists is $O(n)$. Thus, the algorithm runs in time $O(m+n)$.

3.1 Intuition behind the upper bound

The reader may wonder about the origin of the upper bound on the optimal solution used in the previous section. In fact, there is nothing magical about this. It is closely related to the dual of the LP formulation used in Section 2. In fact, the dual of LP (1)-(5) is the following:

$$\min \gamma \tag{7}$$

$$\forall ij \in E \qquad \alpha_{ij} + \beta_{ij} \geq 1 \tag{8}$$

$$\forall i \qquad \gamma \geq \sum_j \alpha_{ij} + \sum_j \beta_{ji} \tag{9}$$

$$\alpha_{ij}, \gamma \geq 0 \tag{10}$$

The upper bound constructed corresponds to a dual solution where α_{ij}, β_{ij} are 0-1 variables. $\alpha_{ij} = 1$ corresponds to the edge ij being assigned to i and $\beta_{ij} = 1$ corresponds to the edge ij being assigned to j. Then γ corresponds to $d^{\max}$. In effect, our proof constructs a dual solution as the greedy algorithm executes. The value of the dual solution is $d^{\max}$, the upper bound in Lemma 3.

We now proceed to the problem of computing $d(G)$ for directed graphs G. Here, the ideas developed in the algorithms for $f(G)$ for undirected graphs G will turn out to be very useful.

4 Exact Algorithm for $d(G)$

Recall that $d(G)$ is the maximum value of $d(S,T)$ over all subsets S,T of vertices. We first present a linear programming relaxation for $d(G)$. Our LP relaxation depends on the value of $|S|/|T|$ for the pair S,T that maximizes $d(S,T)$. Of course, we do not know this ratio a priori, so we write a separate LP for every possible value of this ratio. Note that there are $O(n^2)$ possible values. For $|S|/|T| = c$, we use the following LP relaxation $LP(c)$.

$$\max \sum_{ij} x_{ij} \tag{11}$$

$$\forall ij \qquad x_{ij} \leq s_i \tag{12}$$

$$\forall ij \qquad x_{ij} \leq t_j \tag{13}$$

$$\sum_i s_i \leq \sqrt{c} \tag{14}$$

$$\sum_j t_j \leq \frac{1}{\sqrt{c}} \tag{15}$$

$$x_{ij}, s_i, t_j \geq 0 \tag{16}$$

We now prove the analogue of Lemma 1 earlier.

Lemma 5. *Consider $S,T \subseteq V$. Let $c = |S|/|T|$. then the optimal value of $LP(c)$ is at least $d(S,T)$.*

Proof. We will give a feasible solution $(\bar{x}, \bar{s}, \bar{t})$ for $LP(c)$ (11)-(16) with value $d(S,T)$. Let $x = \frac{\sqrt{c}}{|S|} = \frac{1}{\sqrt{c}\cdot|T|}$. For each $i \in S$, set $\bar{s}_i = x$. For each $j \in T$, set $\bar{t}_j = x$. For each $ij \in E(S,T)$, set $\bar{x}_{ij} = x$. All the remaining variables are set to 0. Now, $\sum_i \bar{s}_i = |S| \cdot x = \sqrt{c}$ and $\sum_j \bar{t}_j = |T| \cdot x = 1/\sqrt{c}$. Thus, this is a feasible solution to $LP(c)$. The value of this solution is

$$|E(S,T)| \cdot x = \frac{|E(S,T)|}{\sqrt{c} \cdot |T|} = \frac{|E(S,T)|}{\sqrt{|S||T|}} = d(S,T)$$

This proves the lemma.

The following lemma is the analogue of Lemma 2.

Lemma 6. *Given a feasible solution of $LP(c)$ with value v we can construct $S,T \subseteq V$ such that $d(S,T) \geq v$.*

Proof. Consider a feasible solution $(\bar{x}, \bar{s}, \bar{t})$ to $LP(c)$ (11)-(16). Without loss of generality, we can assume that for all ij, $\bar{x}_{ij} = \min(\bar{s}_i, \bar{t}_j)$.

We define a collection of sets S,T indexed by a parameter $r \geq 0$. Let $S(r) = \{i : \bar{s}_i \geq r\}$, $T(r) = \{j : \bar{t}_j \geq r\}$ and $E(r) = \{ij : \bar{x}_{ij} \geq r\}$. Since $\bar{x}_{ij} \leq \bar{s}_i$ and $\bar{x}_{ij} \leq \bar{t}_j$, $ij \in E(r) \Rightarrow i \in S(r), j \in T(r)$. Also since $\bar{x}_{ij} = \min(\bar{s}_i, \bar{t}_j)$. Thus $E(r)$ is precisely the set of edges that go from $S(r)$ to $T(r)$.

Now, $\int_0^\infty |S(r)|dr = \sum_i \bar{s}_i \leq \sqrt{c}$. Also, $\int_0^\infty |T(r)|dr = \sum_j \bar{t}_j \leq \frac{1}{\sqrt{c}}$. By the Schwarz inequality,

$$\int_0^\infty \sqrt{|S(r)||T(r)|}dr \leq \sqrt{\left(\int_0^\infty |S(r)|dr\right)\left(\int_0^\infty |T(r)|dr\right)} \leq 1$$

Note that $\int_0^\infty |E(r)|dr = \sum_{ij} \bar{x}_{ij}$. This is the objective function value of the solution. Let this value be v.

We claim that there exists r such that $|E(r)|/\sqrt{|S(r)||T(r)|} \geq v$. Suppose there were no such r. Then

$$\int_0^\infty |E(r)|dr < v \int_0^\infty \sqrt{|S(r)||T(r)|}dr \leq v.$$

This gives a contradiction. To find such an r, notice that we can check all combinatorially distinct sets $S(r), T(r)$ by simply checking $S(r), T(r)$ obtained by setting $r = \bar{s}_i$ and $r = \bar{t}_j$ for every $i \in V$, $j \in V$.

Note that the pair of sets S, T guaranteed by the above proof need not satisfy $|S|/|T| = c$. Putting Lemmas 5 and 6 together, we obtain the following theorem.

Theorem 3.

$$\max_{S,T \subseteq V} \{d(S,T)\} = \max_c \{OPT(LP(c))\} \tag{17}$$

where $OPT(LP(c))$ denotes the value of the optimal solution to $LP(c)$. Further, sets S, T maximizing $d(S,T)$ can be computed from the optimal solutions to the set of linear programs $LP(c)$.

Proof. First we establish the equality (17). From Lemma 5, the RHS $\geq$ the LHS. (Consider the S, T that maximize $d(S,T)$). From Lemma 6, the LHS $\geq$ the RHS. (Set c to be the value that maximizes $OPT(LP(c))$ and consider the optimal solution to $LP(c)$.) The proof of Lemma 6 gives a construction of sets S, T maximizing $d(S,T)$ from the LP solution that maximizes $OPT(LP(c))$.

Remark 1. Note that the proposed algorithm involves solving $O(n^2)$ LPs, one for each possible value of the ratio $c = |S|/|T|$. In fact, this ratio can be guessed to within a $(1+\epsilon)$ factor by using only $O(\frac{\log n}{\epsilon})$ values. It is not very difficult to show that this would yield a $(1+\epsilon)$ approximation. Lemma 5 can be modified to incorporate the $(1+\epsilon)$ factor.

5 Approximation algorithm for $d(G)$

5.1 Intuition behind algorithm

Drawing from the insights gained in analyzing the greedy algorithm for approximating $f(G)$, examining the dual of the LP formulation for $d(G)$ should give us some pointers about how a greedy algorithm for $d(G)$ should be constructed and analyzed.

The dual of $LP(c)$ is the following linear program:

$$\min \sqrt{c} \cdot \gamma + \frac{\delta}{\sqrt{c}} \tag{18}$$

$$\forall ij \quad \alpha_{ij} + \beta_{ij} \geq 1 \tag{19}$$

$$\forall i \quad \gamma \geq \sum_j \alpha_{ij} \tag{20}$$

$$\forall j \quad \delta \geq \sum_i \alpha_{ij} \tag{21}$$

$$\alpha_{ij}, \gamma, \delta \geq 0 \tag{22}$$

Any feasible solution to the dual is an upper bound on the integral solution. This naturally suggests an upper bound corresponding to a dual solution where α_{ij}, β_{ij} are 0-1 variables. $\alpha_{ij} = 1$ corresponds to the edge ij being assigned to i and $\beta_{ij} = 1$ corresponds to the edge ij being assigned to j. Then γ is the maximum number of edges ij assigned to any vertex i (maximum out-degree). δ is the maximum number of edges ij assigned to a vertex j (maximum in-degree). Then the value of the dual solution $\sqrt{c} \cdot \gamma + \frac{\delta}{\sqrt{c}}$ is an upper bound on the $d(S,T)$ for all pairs of sets S, T such that $|S|/|T| = c$.

5.2 Greedy approximation algorithm

We will now use the insights gained from examining the dual of $LP(c)$ to construct and analyze a greedy approximation algorithm. As in the exact algorithm, we need to guess the value of $c = |S|/|T|$. For each such value of c, we run a greedy algorithm. The best pair S, T (i.e. one that maximizes $d(S,T)$) produced by all such greedy algorithms is the output of our algorithm.

We now describe the greedy algorithm for a specific value of c. The algorithm maintains two sets S and T and at each stage removes either the minimum degree vertex in S or the minimum degree vertex in T according to a certain rule. (Here the degree of a vertex i in S is the number of edges from i to T. The degree of a vertex j in T is similarly defined).

1. Initially, $S \leftarrow V, T \leftarrow V$.
2. Let $i_{\min}$ be the vertex $i \in S$ that minimizes $|E(\{i\}, T)|$. Let $d_S \leftarrow |E(\{i_{\min}\}, T)|$.
3. Let $j_{\min}$ be the vertex $j \in T$ that minimizes $|E(S, \{j\})|$. Let $d_T \leftarrow |E(S, \{j_{\min}\})|$.
4. If $\sqrt{c} \cdot d_S \leq \frac{1}{\sqrt{c}} \cdot d_T$ then
   set $S \leftarrow S - \{i_{\min}\}$
   else set $T \leftarrow T - \{j_{\min}\}$.
5. If both S and T are non-empty, go back to Step 2.

Of all the sets S, T produced during the execution of the above algorithm, the pair maximizing $d(S,T)$ is returned as the output of the algorithm.

In order to analyze the algorithm, we produce an upper bound on the optimal solution. The upper bound has the following form suggested by the dual of $LP(c)$: We assign each (directed) edge ij to either i or j. For a vertex i, $d_{out}(i)$ is the number of edges ij' assigned to i. For a vertex j, $d_{in}(j)$ is the number of edges $i'j$ assigned to j. Let $d_{out}^{\max} = \max_i\{d_{out}(i)\}$ and $d_{in}^{\max} = \max_j\{d_{in}(j)\}$. The following lemma gives the upper bound on $d(S,T)$ for all pairs S,T such that $|S|/|T| = c$ in terms of $d_{out}^{\max}$ and $d_{in}^{\max}$.

Lemma 7.

$$\max_{|S|/|T|=c}\{d(S,T)\} \leq \sqrt{c} \cdot d_{out}^{\max} + \frac{1}{\sqrt{c}} \cdot d_{in}^{\max}$$

Proof. This follows directly from the fact that the assignment of edges to vertices corresponds to a 0-1 solution of the dual to $LP(c)$. Note that the value of the corresponding dual solution is exactly $\sqrt{c} \cdot d_{out}^{\max} + \frac{1}{\sqrt{c}} \cdot d_{in}^{\max}$. We give an alternate, combinatorial proof of this fact.

Consider the pair of sets S,T that maximizes $d(S,T)$ over all pairs S,T such that $|S|/|T| = c$. Now, each edge in $E(S,T)$ must be assigned to a vertex in S or a vertex in T. Thus

$$\begin{aligned}
|E(S,T)| &\leq |S| \cdot d_{out}^{\max} + |T| \cdot d_{in}^{\max} \\
d(S,T) = \frac{|E(S,T)|}{\sqrt{|S||T|}} &\leq \sqrt{\frac{|S|}{|T|}} d_{out}^{\max} + \sqrt{\frac{|T|}{|S|}} d_{in}^{\max} \\
&= \sqrt{c} \cdot d_{out}^{\max} + \frac{1}{\sqrt{c}} \cdot d_{in}^{\max}
\end{aligned}$$

Now, the assignment of edges to one of the end points is constructed as the algorithm executes. Note that a separate assignment is obtained for each different value of c. Initially, all edges are unassigned. When a vertex is deleted from either S or T in Step 4, the vertex is assigned all edges that go from the vertex to the other set (i.e. if $i_{\min}$ is deleted, it gets assigned all edges from $i_{\min}$ to T and similarly if $j_{\min}$ is deleted). We maintain the invariant that all edges that go from the current set S to the current set T are unassigned; all other edges are assigned. At the end of the execution of the algorithm, all edges are assigned.

Let $d_{out}^{\max}$ and $d_{in}^{\max}$ be defined as before for the specific assignment constructed corresponding to the execution of the greedy algorithm. The following lemma relates the value of the solution constructed by the greedy algorithm to $d_{out}^{\max}$ and $d_{in}^{\max}$.

Lemma 8. *Let v be the maximum value of $d(S,T)$ for all pairs of sets S,T obtained during the execution of the greedy algorithm for a particular value of c. Then $\sqrt{c} \cdot d_{out}^{\max} \leq v$ and $\frac{1}{\sqrt{c}} \cdot d_{in}^{\max} \leq v$.*

Proof. Consider an execution of Steps 2 to 5 at any point in the algorithm. Since $i_{\min}$ is selected to be the minimum degree vertex in S, its degree is at most $|E(S,T)|/|S|$, i.e. $d_S \leq |E(S,T)|/|S|$. Similarly $d_T \leq |E(S,T)|/|T|$. Now,

$$\min(\sqrt{c} \cdot d_S, \frac{1}{\sqrt{c}} d_T) \leq \sqrt{d_S d_T} \leq \frac{|E(S,T)|}{\sqrt{|S||T|}} \leq v.$$

If $\sqrt{c} \cdot d_S \leq \frac{1}{\sqrt{c}} d_T$, then $i_{\min}$ is deleted and assigned the edges going from $i_{\min}$ to T. In this case, $\sqrt{c} \cdot d_S \leq v$. If this were not the case, $j_{\min}$ is deleted and assigned edges going from S to $j_{\min}$. In this case, $\frac{1}{\sqrt{c}} d_T \leq v$. Note that a particular vertex gets assigned edges to it only if it is removed from either S or T in Step 4. This proves that $\sqrt{c} \cdot d_{out}^{\max} \leq v$ and $\frac{1}{\sqrt{c}} d_{in}^{\max} \leq v$.

Putting Lemmas 7 and 8 together, we get the following.

Lemma 9. *Let v be the maximum value of $d(S,T)$ for all pairs of sets S,T obtained during the execution of the greedy algorithm for a particular value of c. Then,*

$$v \geq \frac{1}{2} \max_{|S|/|T|=c} \{d(S,T)\}.$$

Observe that the maximizing pair S,T in the above lemma need not satisfy $|S|/|T| = c$.

The output of the algorithm is the best pair S,T produced by the greedy algorithm over all executions (for different values of c). Let S^*, T^* be the sets that maximize $d(S,T)$ over all pairs S,T. Applying the previous lemma for the specific value $c = |S^*|/|T^*|$, we get the following bound on the approximation ratio of the algorithm.

Theorem 4. *The greedy algorithm gives a 2 approximation for $d(G)$.*

Remark 2. As in the exact LP based algorithm, instead of running the algorithm for all $\Omega(n^2)$ values of c, we can guess the value of c in the optimal solution to within a $(1+\epsilon)$ factor by using only $O(\frac{\log n}{\epsilon})$ values. It is not very difficult to show that this would lose only a $(1+\epsilon)$ factor in the approximation ratio. We need to modify Lemma 7 to incorporate the $(1+\epsilon)$ factor.

Running Time Similar to the implementation of the greedy algorithm for $f(G)$, the greedy algorithm for $d(G)$ *for a particular value of* c can be implemented to run in $O(m+n)$ time. By the above remark, we need to run the greedy algorithm for $O(\frac{\log n}{\epsilon})$ values of c in order to get a $2+\epsilon$ approximation.

6 Conclusion

All the algorithms presented in this paper generalize to the setting where edges have weights. In the weighted setting, the linear time implementation does not carry over, since it depends on the fact that vertex degrees are integers bounded by n. However, the algorithms can be implemented using Fibonacci heaps to determine the minimum degree vertex in every iteration. Both the greedy algorithm for $f(G)$ as well as the greedy algorithm for $d(G)$ (for a single value of c) run in $O(m + n \log n)$ time in this case.

In conclusion, we mention some interesting directions for future work. In the definition of density $d(G)$ for directed graphs, the sets S, T were not required to be disjoint. What is the complexity of computing a slightly modified notion of density $d'(G)$ where we maximize $d(S, T)$ over disjoint sets S, T ? Note that any α-approximation algorithm for $d(G)$ can be used to obtain an $O(\alpha)$-approximation for $d'(G)$. Finally, it would be interesting to obtain a flow based algorithm for computing $d(G)$ exactly, along the same lines as the flow based algorithm for computing $f(G)$.

7 Acknowledgments

I would like to thank Ravi Kannan for introducing me to the problem and giving me a preliminary version of [9]. I would also like to thank Baruch Schieber for suggesting an improvement to the algorithm in Section 5. The previous inelegant version had a worse approximation guarantee. I thank Samir Khuller for suggesting the linear time implementation of the greedy algorithm for unweighted graphs.

References

1. Y. Asahiro and K. Iwama. Finding Dense Subgraphs. *Proc. 6th International Symposium on Algorithms and Computation (ISAAC)*, LNCS 1004, 102–111 (1995).
2. Y. Asahiro, K. Iwama, H. Tamaki and T.Tokuyama. Greedily Finding a Dense Subgraph. *Journal of Algorithms*, 34(2):203–221 (2000).
3. P. Drineas, A. Frieze, R. Kannan, S. Vempala and V. Vinay. Clustering in Large Graphs and Matrices. *Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, 291–299 (1999).
4. U. Feige, G. Kortsarz and D. Peleg. The Dense k-Subgraph Problem. *Algorithmica*, to appear. Preliminary version in *Proc. 34th Annual IEEE Symposium on Foundations of Computer Science*, 692–701 (1993).
5. U. Feige and M. Seltser. On the Densest k-Subgraph Problem. Weizmann Institute Technical Report CS 97-16 (1997).
6. A. Frieze, R. Kannan and S. Vempala. Fast Monte-Carlo Algorithms for Finding Low Rank Approximations. *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science*, 370–378 (1998).
7. G. Gallo, M. D. Grigoriadis, and R. Tarjan. A Fast Parametric Maximum Flow Algorithm and Applications. *SIAM J. on Comput.*, 18:30–55 (1989).
8. D. Gibson, J. Kleinberg and P. Raghavan. Inferring web communities from Web topology. *Proc. HYPERTEXT*, 225–234 (1998).
9. R. Kannan and V. Vinay. Analyzing the Structure of Large Graphs. *manuscript*, August 1999.
10. J. Kleinberg. Authoritative sources in hypertext linked environments. *Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, 668–677 (1998).

11. J. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan and A. Tomkins. The web as a graph : measurements, models, and methods. *Proc. 5th Annual International Conference on Computing and Combinatorics (COCOON)*, 1–17 (1999).
12. S. R. Kumar, P. Raghavan, S. Rajagopalan and A. Tomkins. Trawling Emerging Cyber-Communities Automatically. *Proc. 8th WWW Conference*, Computer Networks, 31(11–16):1481–1493, (1999).
13. E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston (1976).