The Open University of Israel
Department of Mathematics and Computer Science

Approximating Graph Density Problems

Thesis submitted as partial fulfillment of the requirements
towards an M.Sc. degree in Computer Science
The Open University of Israel
Computer Science Division

By
Elazar Leibovich

Prepared under the supervision of Prof. Zeev Nutov

September 2009

Abstract

The density of a graph $G = (V, E)$ with edge capacities $\{c(e) : e \in E\}$ is $c(E)/|V|$. Finding dense components in graphs is a major topic in graph theory algorithms. The most basic problem is the Densest Subgraph problem, that seeks a subgraph of maximum density. This problem can be solved in polynomial time, and we will give a systematic description of several polynomial time algorithms for this problem.

We then consider maximum density problems of the following type. Given a graph $G = (V, E)$ with edge capacities $\{c(e) : e \in E\}$ and a property Π, find the densest subgraph $G' \subseteq G$ that satisfies property Π. Here the most famous example is the Densest k-Subgraph problem, where the property Π is "G' has exactly k nodes". The Densest k-Subgraph problem is strongly NP-hard, and the best approximation ratio known for it is $O(n^{1/4+\varepsilon})$.

We consider two other density problems: the Densest Path problem where G' is required to be a path, and the Densest Tree problems where G' is required to be a tree. These problems are shown to be NP-hard. We describe a general framework to attack some density problems based on decompositions of feasible solutions, and use it to derive a PTAS for the the Densest Path and the Densest Tree problems.

Acknowledgement

I wish to thank my thesis advisor, Prof. Zeev Nutov, for his friendly, knowledgeable and invaluable guidance, advice and support throughout the gestation and development of the results presented in this thesis.

Contents

1 Introduction

1.1 Graph density problems

The *density* of a graph $G = (V, E)$ with edge capacities $\{c(e) : e \in E\}$ is $\sigma(G) = c(E)/|V|$. Given a graph property Π let $\Pi(G)$ denote the family of subgraphs of G that satisfy Π. We consider the following type of problems:

Densest Π-Subgraph:
Instance: A graph $G = (V, E)$ with edge capacities $\{c(e) : e \in E\}$ and a family $\Pi(G)$ of subgraphs of G.
Objective: Find a subgraph $G' \in \Pi(G)$ of maximum density.

The Π-Maximization problem is to maximize $c(E')$ over all $G' = (V', E') \in \Pi(G)$. The family $\Pi(G)$ may have size exponential in the size of G; it does not have to be given explicitly, but might be specified by some condition, and we may require that certain queries related to $\Pi(G)$ can be answered in polynomial time. E.g., $\Pi(G)$ can be the family of paths, or of trees, giving the density problems Densest Path, or Densest Tree, respectively. The corresponding maximization problems, where one search for the set with maximum edge's total weight and not for maximum edge/vertex ratio, are Longest Path, or Maximum Tree. In general, density problems often arise in greedy methods in approximation algorithms, c.f., [10, 13]. The most famous density problem is the Densest k-Subgraph problem. Here the property Π is "the graph has k nodes" for a given integer k. This problem is NP-hard for unit capacities as it includes the Maximum Clique problem. Indeed, given an instance G of Maximum Clique, the density of any k-subgraph is $\leq (k-1)/2$, and a k-subgraph has density $(k-1)/2$ if, and only if, it is a clique. The best ratio known for this problem is $O(n^{1/4+\varepsilon})$ [2] (see also the previous $O(n^{1/3})$-approximation in [11]). We will mainly consider the following three problems:

Densest Subgraph: Here the family $\Pi(G)$ consists of all subgraph of G. Several polynomial time algorithms for this problem are known, see Section 2.

Densest Path: Here the property Π is "the graph is a path". This problem is NP-hard as it includes the Hamiltonian Path problem. Indeed, given an instance G of Hamiltonian Path, the density of any path is $\leq (n-1)/n$, and a path has density $(n-1)/n$ if, and only if, it is Hamiltonian. We will show that this problem admits a PTAS, see Section 3.

Densest Tree: Here property Π is "the graph is a tree". We will show that this problem is NP-hard and admits a PTAS, see Section 3.

1.2 This Thesis

In this thesis, we survey some algorithms for the Densest Subgraph problem. We then develop a generic framework for approximating a certain type of "decomposable" problems. We apply this approach for the Densest Path and Densest Tree problems to obtain the following result.

Theorem 1.1 Densest Path *and* Densest Tree *admit a PTAS. Furthermore, these problems are NP-hard.*

1.3 Some previous work

There are several applications for finding dense components in a graph. Charikar [6] mentions an application of the Densest Subgraph problem for finding dense components in the Internet. There is an algorithm described in [15] called HITS which starts from a basic set of web pages which are known to be authorative on a particular topic, and a set of web pages which are known to contain resources about the topics (these are called *authorities* and *hubs* respectively). Additional resources are recognized by number of links they have to the authorities, additional authorities pages are recognized by the number of links they have from the hubs. Dense components on the Internet are also useful in the context of *link-farms*. *Link-farms*, or *link-spams* are websites that attempt to manipulate search engine rankings through aggressive interlinking to simulate popular content. Those link-farms are being dense subgraph in the big graph of the Internet. Gibson et al [16] describes an algorithm for finding link-farms in the internet. Miles and Phoha [23] show how to utilize a bipartitie clique to analyze a user search patterns. Looking at graph formed by the web pages the user has viewed, one can find a bipartite-clique. The part of web pages with linking to another set of web pages would be recognized as *referers*, and the other part would be recognized as the *request*. Obviously, the densest bipartite subgraph is a relaxation of the bipartite clique. Buehrer et al. [5] introduce a way to compress websites related graph data, while still allowing random access to the compressed data. His solution is based on finding *communities* in the web-graph, which are actually dense components.

The most studied density problem is the Densest k-Subgraph problem, which is strongly NP-hard. The best known approximation for Densest k-Subgraph is $\Omega(n^{1/4+\varepsilon})$ [2]. Furthermore, many well known problems can be reduced to Densest k-Subgraph, and this sometimes

serves as evidence that a polylogarithmic approximation for the problem may not exists, or at least hard to obtain. Some other interesting result appear in [11]. The approximation gap for Densest k-Subgraph is quite large, as the best hardness of approximation result for it is ruling out PTAS by Khot [20]. The Densest k-Subgraph problem is known to be NP-hard, even on special classes of graphs. These classes include bipartite graphs, planar graphs [19], split graphs (graphs whose vertices can be partitioned into a clique and an independent set) [8]. It is even hard for graphs with maximum degree of 3 [12]. For chordal graph Liazi et al. [21] shows a 3-approximation. The Densest k-Subgraph problem also admits a PTAS for dense graphs with $\Omega(n^2)$ edges [1]. For the capacitated version on a complete graph when the capacities obey the triangle inequality, the problem admits a 2-approximation [17].

Huang and Kahng [18] studied the Densest Sub-hypergraph problem; here given a hypergraph $H = (V, E)$, the density of the sub-hypergraph induced by a subset $V' \subseteq V$ of nodes is the number of hyperedges in E that are completely contained in V', divided by $|V'|$. They show that the Densest Sub-hypergraph problem can be solved in polynomial time.

Several papers considered problems of finding a subgraph of *minimum* density. The problem of finding in a tree a minimum density subtree with exactly k edges can be solved in polynomial time, see the works of Maffioli [22], Blum and Ehrgott [4], and Blum [3]. Chekuri and Korula [7] considered the problem of finding a minimum density 2-connected subgraph, and gave an $O(\log n)$-approximation algorithm for this problem. However, we are not aware of any approximation results for the (maximization) Densest Tree or Densest Path in general graphs.

2 Algorithms for the Densest Subgraph problem

2.1 The (f, p)-Densest Set problem

Density problems in general are tightly connected to the general fractional programming problem. Generally speaking, given a set S, and two functions $f, g : S \to \mathbb{R}$, the fractional programming minimization/maximization problem is to find $x^* \in S$ which maximizes/minimizes the function

$$h(x) = \frac{f(x)}{g(x)} \ . \tag{1}$$

See [9] for a more elaborated introduction. The most efficient algorithm that solves a fractional programming problem, depends on the type of the set S in the problem. There are various algorithms for many types of sets.

The essence of the method we introduce here is described in Gallo et al. [14]. They casted the problem of the densest subgraph to a fractional programming problem. Then, they used maximum flow algorithms, in order to solve the fractional programming problem.

Here, we will cast the densest subgraph problem to a fractional programming programming problem in the same way it was done in [14]. However we will show a more general technique to maximize the linear program, a technique applicable to all density problems whose weight function is supermodular.

Recall that the density of a graph $G = (V, E)$ with edge capacities $\{c(e) : e \in E\}$ is $\sigma(G) = c(E)/|V|$. In a more general setting, we might also be given node weights $\{w(v) : v \in V\}$; in this case $\sigma(G') = c(E')/w(V')$.

Densest Π-Subgraph is a particular case of the following generic fractional programming problem. Let f, p be integral set functions on a groundset U given by an evaluation oracle and bounded by a polynomial in $|U|$, so that $p(X) > 0$ if $X \neq \emptyset$. The (f, p)-*density* of $X \subseteq U$ is $f(X)/p(X)$. The (f, p)-Densest Set problem is to find a nonempty set of maximum (f, p)-density.

Example 1. The Densest Subgraph problem can be casted as (f, p)-Densest Set as follows. The groundset is $U = V$. The function f is defined as $f(V') = |E(V')|$, that is, the number of edges in the subgraph of G induced by V'. The function p is defined as $p(V') = |V'|$.

Example 2. The Densest k-Subgraph problem can be casted as (f, p)-Densest Set as follows. The groundset is $U = V$. The function f is defined as $f(V') = |E(V')|$ if $|V'| = k$ and $f(V') = 0$ otherwise. The function p is defined as $p(V') = |V'|$.

Example 3. The Densest Tree problem can be casted as (f,p)-Densest Set as follows. The groundset is $U = V$ and $p(V') = |V'|$ for all $V' \subseteq V$. The function f is defined as follows: if the subgraph $G' = (V', E(V'))$ induced by V' in G is connected then $f(V')$ is the maximum capacity of a spanning tree in G'; otherwise, $f(V') = 0$.

We have seen that solving the (f,p)-densest set would allow us to solve the Densest Subgraph problem, we will thus devise a method to solve the (f,p)-densest set problem.

Theorem 2.1 (f,p)-Densest Set *can be solved in polynomial time if* $\max_{\emptyset \neq X \subseteq V} f(X)$ *and* $\min_{\emptyset \neq X \subseteq U} p(X)$ *can be computed in polynomial time, and if for any rational number* q *we can compute in polynomial time the maximizer of the function*

$$h(q) \equiv \max_{\emptyset \neq X \subseteq U} (f(X) - qp(X)) = \max_{\emptyset \neq X \subseteq U} p(X) \left[f(X)/p(X) - q \right] \ . \tag{2}$$

Proof: Note that h is a monotone (strictly) decreasing function, since for any $\emptyset \neq X \subseteq U$ and any $\varepsilon > 0$ we have:

$$p(X) \left[f(X)/p(X) - q \right] > p(X) \left[f(X)/p(X) - (q - \varepsilon) \right]$$

This is true since p is non-negative.

Let $q^* = \max_{\emptyset \neq X \subseteq U} f(X)/p(X)$ and let $M = \max_{\emptyset \neq X \subseteq V} f(X)$ and $\mu = \min_{\emptyset \neq X \subseteq U} p(X)$. Then $h(q) = 0$ if and only if $q = q^*$, $h(q) < 0$ for $q < q*$, $h(q) > 0$ for $q > q*$. Also, $0 \leq q^* \leq M/\mu$. As $h(q)$ is monotone decreasing, we can apply binary search in the range $0 \leq q \leq M/\mu$ to compute q^*, as well as the corresponding set X^*. □

Definition 2.1 *A set function* f *is* supermodular *if* $f(X) + f(Y) \leq f(X \cap Y) + f(X \cup Y)$ *for all* $X, Y \subseteq U$*;* f *is* submodular *if* $-f$ *is supermodular.*

The problem of maximizing a supermodular function can be solved in polynomial time, assuming that the function is given by a (polynomial time) evaluation oracle, see for example [25]. Thus Theorem 2.1 implies:

Claim 2.2 (f,p)-Densest Set *can be solved in polynomial time if the set function* $f - qp$ *is supermodular for any* $q \geq 0$*.*

Corollary 2.3 *Let* f *be supermodular and* p *a submodular functions defined over the same groundset* U*. Then* $h(q) = f(X) - qp(X)$ *is supermodular for all* $q \geq 0$*. Thus* (f,p)-Densest Set *can be solved in polynomial time in this case.*

Proof: It is easy to see that since p is submodular, so is qp for any $q \geq 0$. Hence $-qp$ is supermodular. Therefore h is a sum of two supermodular functions, which easily implies that h is also supermodular. The last statement follows from Claim 2.2. □

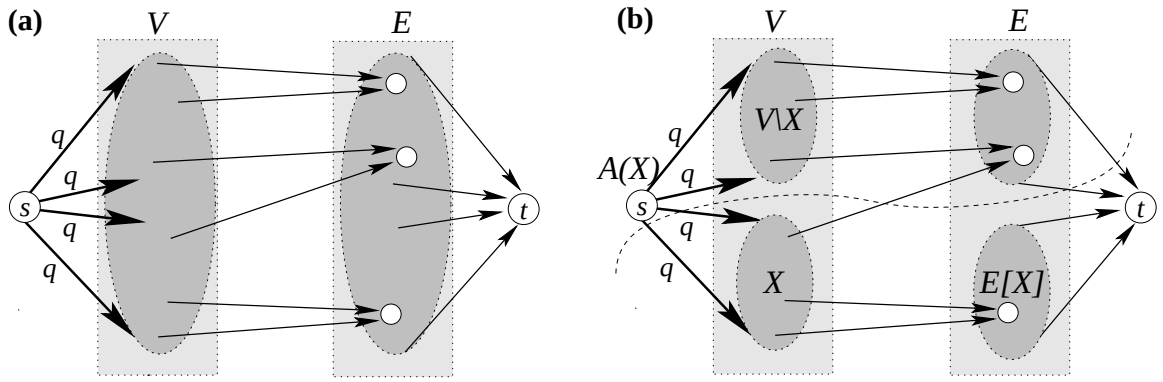

Figure 1: (a) The flow network J. (b) The cut $A(X)$.

It is well known that the function $\gamma(X) = |E(X)|$ is supermodular, c.f., [25]. As for $f(X) = |E(X)|$ and $p(X) = |X|$ we get the Densest Subgraph problem, Corollary 2.3 implies a polynomial time algorithm for Densest Subgraph.

2.2 Solving Densest Subgraph via flows

Here we will present a more efficient algorithm for the Densest Subgraph problem using max-flows. The idea is again to show an algorithm for computing $h(q)$, while finding an inclusion maximal set X for which the maximum in (2) is attained. This is done by constructing a flow-network J, and finding the max-flow/min-cut in this network.

The construction of the flow network J as follows, see Figure 1(a).

1. Let $H = (U_1, U_2, F)$ be the *adjacency graph* of $G(V, E)$, namely, H is a bipartite graph with two parts U_V,U_E. For every vertex v in G, we will have a vertex u_v in U_V, for every edge e in G, we will have a vertex w_e in U_E. There's an edge between u_v to w_e if and only if v is an endpoint of e in G. Note that a simple way to obtain H from G is to subdivide every edge of G by a node.

2. Let J be a flow network obtained from H as follows.

   (a) Assign capacity 1 to the edges of H and direct them from U_V to U_E.

   (b) Add a source s and an edge sv of capacity q for every $v \in U_V$.

   (c) Add a sink t and an edge et of capacity 1 for every $w \in U_E$.

Let A be a subset of nodes of J. Let $\delta_J(A)$ be the set of edges in J leaving A, namely,

with tail in A and head not in A. We say that A is an st-cut if $s \in A$ and $t \notin A$. The capacity of the cut A is the sum of the capacities of the edges in $\delta_J(A)$.

The algorithm for computing $h(q)$ is as follows.

1. Constructs the network J as above.
2. Return the minimum cut of maximum cardinality in J.

It is known that step 2 can be implemented in $O(|E| \log |V|(|E| + |V| \log |E|))$ time, c.f. [24]. Hence the algorithm can be implemented in polynomial time.

We will prove that the weight of the minimum cut of maximum cardinality in J is $h(q)$.

Claim 2.4 *For $X \subseteq V$ in G, let $A(X) = \{s\} \cup (U_V \setminus X) \cup (U_E \setminus E(X))$. Then $A(X)$ is an st-cut of capacity $|E| - (|E(X)| - q \cdot |X|)$.*

Proof: $A(X)$ is an st-cut in J since $s \in A(X)$ and $t \notin A(X)$. To see that the capacity of $A(X)$ is as claimed, consider Figure 1(b). The main observation is that in J there is no edge from $U_V \setminus X$ to $E[X]$, since in G an edge that is incident to a node in $V \setminus X$ is not in $E[X]$. Therefore the edges leaving $A(X)$ are of three types:

- The edges from s to X; there are $|X|$ such edges, of capacity q each.
- The edges from $E \setminus E[X]$ to t; there are $|E| - |E[X]|$ such edges, of capacity 1 each.
- The edges from $U_V \setminus X$ to $|E|$.

Consequently, the capacity of the cut $A(X)$ is

$$\begin{aligned} c(A(X)) = q \cdot |X| + cut(X, V \setminus X) + |E[V \setminus X]| = \\ q|X| + (|E| - |E[X]|) = \\ |E| - (|E[X]| - q \cdot |X|) \end{aligned}$$

as claimed. □

Claim 2.5 *Let A be a minimum st-cut of maximum cardinality in the network J mentioned above. If $v \in A$ for $v \in V$ then $e \in A$ for every edge e incident to v.*

Proof: In the network J, for every node $e \in E$ (namely, for the node corresponding to e) the following holds:

- Exactly one edge leave e, which is et, and its capacity is 1.
- Exactly two edges enter e, which are ue, ve, where u, v are the endnodes of e in G, and their capacity is 1.

Thus if A is an st-cut in J, and if $v \in A$ for some $v \in V$ but $e \notin A$ for some edge e that is incident to v in G, then $A \cup \{e\}$ is an st-cut in J of capacity at most the capacity of A. The statement now follows from the maximality of A. □

Lemma 2.6 *Let A be a minimum cut of maximum cardinality in network J as above, Let X_q be the equivalent cut in the original graph G we have that*

$$h(q) = |E[X_q]| - q \cdot |X_q|$$

.

Proof: From Claim 2.4 we have that $e \in E \setminus A$ if, and only if, in J the tails of the two edge entering e are in $X_q = V \setminus A$. This is equivalent to $E \setminus A = E[X_q]$, namely, $A = \{s\} \cup (V \setminus X_q) \cup (E - E[X_q])$. Now observe that

$$\begin{aligned} h(q) = \max_{\emptyset \neq X \subseteq V} (|E[X]| - q \cdot |X|) = \\ |E| - \min_{\emptyset \neq X \subseteq V} (|E| - (|E[X]| - q \cdot |X|)) = \\ |E| - \min_{\emptyset \neq X \subseteq V} c(A(X)) \ . \end{aligned}$$

Thus the maximum of $h(q) = \max_{\emptyset \neq X \subseteq V}(|E[X]| - q \cdot |X|)$ and the minimum of $\min_{\emptyset \neq X \subseteq V} c(A(X))$ is attained for the same node set X. Since the cut $A = A(X_q)$ is a minimum st-cut, the minimum of $\min_{\emptyset \neq X \subseteq V} c(A(X))$ is attained for $X = X_q$. Consequently, $h(q) = |E[X]| - q \cdot |X|$, as claimed.

Note, that this lemma holds even if there are many minimal cuts of maximum cardinality since the weight of all those cuts would still be equal to $h(q)$. □

The last claim implies a polynomial time algorithm for computing $h(q)$ for any $q \geq 0$. A polynomial time algorithm for Densest Subgraph now follows from Theorem 2.1.

3 Densest Path and Densest Tree problems

3.1 General framework

Let us introduce a general method for approximating problems which are decomposable. This method is applicable for both for the Densest Path and the Densest Tree problems. In particular, we will deduce the first part of Theorem 1.1.

Lemma 3.1 *Suppose that for an instance $G, c, \Pi(G)$ of* Π-Densest Subgraph *the following holds. For every $H = (U, F) \in \Pi(G)$ and any integer k there exists a non-empty family $\mathcal{F} = \{(U_1, F_1), (U_2, F_2), \ldots\} \subseteq \Pi(H)$ so that:*

1. *$|U_i| \leq k$ for every i.*
2. *$|U| \geq \beta \cdot \sum_i |U_i|$.*
3. *$c(F) \leq \gamma \cdot \sum_i c(F_i)$.*

Then, if Densest Π-Subgraph *(resp., if* Π-Maximization*) admits a ρ-approximation in $T(n)$ time, then* Densest Π-Subgraph *can be approximated within $\rho \cdot (\beta/\gamma)$ in $S(n,k) \cdot T(k) = O(n^k) \cdot T(k)$ time, where $S(n,k) = \binom{n}{k} = O(n^k)$ (resp., $S(n,k) = \sum_{i=1}^{k} \binom{n}{k}$).*

Proof: The algorithm computes a ρ-approximation for Densest Π-Subgraph (resp., for Π-Maximization) on $G[X]$ for all $\{X \subseteq V : |X| = k\}$ (resp., for all $\{X \subseteq V : |X| \leq k\}$. Then, among the subgraphs computed, it returns the densest one.

Clearly, the time complexity is as claimed. The solution returned is feasible, since there exists X so that Densest Π-Subgraph applied on $G[X]$ has a feasible solution, and since for every X we apply an algorithm that returns a feasible solution, if such exists.

It remains to prove the approximation ratio. This follows by a standard averaging argument. Let $H = (U, F)$ be an optimal solution to the given Densest Π-Subgraph instance, and let $H_i = (U_i, F_i)$ be the decomposition of H as above. We have

$$\mathsf{opt} = \frac{c(F)}{|U|} \leq \frac{\gamma \cdot \sum_i c(F_i)}{\beta \cdot \sum_i |U_i|} \ .$$

Hence there exists an index i so that

$$\frac{c(F_i)}{|U_i|} \geq \frac{\beta}{\gamma} \cdot \mathsf{opt} \ .$$

The statement follows, as the algorithm returns a subgraph of density at least $\rho \cdot c(F_i)/|U_i|$.

□

Note that in the algorithm we can use as a subroutine a (possibly exponential) algorithm for Densest Π-Subgraph or for Π-Maximization. Π-Maximization is used if it is easier than Densest Π-Subgraph, e.g., as in Densest Tree. Otherwise, we may use an exponential time algorithm for Densest Π-Subgraph to obtain an approximation scheme. Another possibility is to use recursion.

We now show how to apply Lemma 3.1 on the problems Densest Tree and Densest Path. For each of these problems we will prove that it is NP-hard.

3.2 Densest Path

PTAS for Densest Path: We will show that Densest Path admits a PTAS with running time $O\left(n^{1/\varepsilon} \cdot (1/\varepsilon)!\right)$. Let $H = (U, F)$ be a path. For any k, we can decompose H into at most $|U|/k$ paths $\{(U_1, F_1), (U_2, F_2), \ldots\}$ of length $\leq k$ each, so that their edge sets partition F, and so that $|U_i \cap U_j|$ is 1 if $|i - j| = 1$ and is 0 otherwise. Hence $c(F) = \sum_i c(F_i)$ and $\sum_i |U_i| \leq |U| + |U|/k$; thus $|U| \geq (1 - 1/k) \sum_i |U_i|$. Π-Maximization can be solved in $O(k!)$ time in this case. Consequently, we can substitute $\beta = 1 - 1/k$, $\gamma = 1$, and $\rho = 1$ in Lemma 3.1. Thus for any k we have a $(1 - 1/k)$-approximation in $O(n^k \cdot k!)$ time. Given $\varepsilon > 0$, we set $k = 1/\varepsilon$, and obtain an approximation scheme as claimed.

NP-hardness of Densest Path: As was mentioned in the Introduction, Densest Path is NP-hard as it includes the Hamiltonian Path problem: given an instance G of Hamiltonian Path, the density of any path is $\leq (n-1)/n$, and a path has density $(n-1)/n$ if, and only if, it is Hamiltonian.

3.3 PTAS for Densest Tree

We will show that Densest Tree admits a PTAS with running time $O\left(n^{1/\varepsilon}(1/\varepsilon)\log(1/\varepsilon)\right)$. Our algorithm is based on the following decomposition.

Lemma 3.2 *Given an integer $\ell \geq 1$, any tree $T = (U, F)$ with at least ℓ edges can be decomposed into a family $\mathcal{T} = \{T_1, \ldots, T_t\}$ of edge disjoint subtrees, $T_i = (U_i, F_i)$, so that each subtree contains at least ℓ edges and at most 3ℓ edges. In particular, every two subtrees have at most one node in common, $t \leq |U|/\ell$, and thus $\sum_i |U_i| \leq |U|(1 + 1/\ell)$.*

Proof: We first prove directly that for $n = 3\ell + 1$ there exists a decomposition as in the lemma into 2 subtrees, each with each with at least ℓ edges and at most 2ℓ edges. For a node r of T, the r-subtrees (of T) are the inclusion maximal subtrees of T containing exactly one edge incident to r. Here we claim that there exists a node r so that every r-subtree has at most 2ℓ edges. To see this, initially root T at an arbitrary node r. If there is an r-subtree with at least $2\ell + 1$ edges, then set $r \leftarrow$ the neighbor of r in this subtree. The maximum number of edges in an r-subtree decreases by at least one, hence eventually we will find a node r as required. Now, if there is an r-subtree with $\ell + 1$ edges, then this r-subtree and the union of all other r-subtrees is a decomposition into 2 parts as required. Else, every r-subtree has at most ℓ edges. Then there exists a collection of r-subtrees so that the total number of edges in them is between ℓ and 2ℓ. Thus the union of these subtrees, and the union of other r-subtrees is a decomposition as required.

Now we prove the statement in the lemma by induction.on the number m of edges of T. If $\ell \leq m \leq 3\ell$ then the statement is trivial. Assume therefore that $m \geq 3\ell + 1$. Let v be a leaf of T and let u be the neighbor of v in T. By the induction hypothesis, $T - \{v\}$ admits a decomposition $\mathcal{T}'$ as in the lemma. Let T_i be the tree in this decomposition that contains u, and let T' be obtained from T_i by adding the node v and the edge uv. As T_i has at most 3ℓ edges, T' has at most $3\ell + 1$ edges. If T' has at most 3ℓ edges, then by replacing T_i by T' in $\mathcal{T}'$ we obtain a decomposition $\mathcal{T}$ as required. Otherwise, T' has exactly $3\ell + 1$ edges, and thus can be decomposed into trees T_1, T_2 each with at least ℓ edges and at most 2ℓ edges. □

Now, we can apply Lemma 3.1 with $k = 3\ell$, $\beta = 1 - 1/(\ell + 1)$, and $\gamma = 1$. The Π-maximization problem in this case is just the maximum spanning tree problem that can be solved in $O(\ell^2 \log \ell)$ time. Let $\varepsilon = 1/\ell$. Then we have a $(1 - \varepsilon)$ approximation in $O(n^{3\ell} \cdot \ell^2 \log \ell) = O(n^{3/\varepsilon} \cdot (1/\varepsilon^2) \log 1/\varepsilon)$ time.

Remark: The same algorithm applies for the Densest Forest problem.

3.4 NP-hardness of Densest Tree

We have already shown before that Densest Path is NP-hard by a simple reduction from Hamiltonian Path. Here we will show that the Densest Tree problem is NP-hard by a more involved reduction from the following known problem:

Steiner Tree:
Instance: A graph $G = (V, E)$ and a subset $U \subseteq V$, $|U| = k$.
Objective: Find a subtree T of G with minimum number of edges that contains U.

Given an instance G, U of Steiner Tree, obtain an instance G', c of Densest Tree as follows. Add to G a copy U' of U and connect every $u' \in U'$ to the corresponding node $u \in U$. The capacities are: 1 for edges in E and $a = 1 + 1/k + 1/k^2$ for the new edges.

Let T' be a subtree of G'. Suppose that T' contains ℓ nodes from U, j nodes from $V - U$, and $\ell' \leq \ell$ nodes from U'. Then the density of T is:

$$\sigma(T) = \frac{a\ell' + \ell + j - 1}{\ell' + \ell + j} < a \ . \tag{3}$$

A subtree T' of G' is *proper* if $u \in T'$ implies $u' \in T'$. For a subtree T of G let T' denote the minimal proper subtree of G' that contains T, namely, T' is obtained from T by adding for every $u \in T$ the node u' and the edge uu'.

Lemma 3.3 *Any optimal densest subtree T' of G' is proper.*

Proof: Otherwise, add to T' one more node from U'. The density of the "added part" is a, while $\sigma(T) < a$ by (3). Hence we obtain a tree of larger density, contradicting the optimality of T'. □

Lemma 3.4 *T is an optimal Steiner tree in G if, and only if, T' is the densest tree in G'.*

Proof: Let T' be an optimal subtree of G'. Then $\ell' = \ell$, by Claim 3.3. Thus by (3):

$$\sigma(T') = \frac{a\ell + \ell + j - 1}{2\ell + j} = 1 + \frac{\ell(a-1) - 1}{2\ell + j} = 1 + \frac{1}{2\ell + j}\left(\frac{\ell}{k} + \frac{\ell}{k^2} - 1\right) \ .$$

We claim that $\ell = k$. Indeed,

$$\frac{\ell}{k} + \frac{\ell}{k^2} - 1 = \frac{1}{k} \ \text{ if } \ \ell = k \ ,$$

$$\frac{\ell}{k} + \frac{\ell}{k^2} - 1 \leq -\frac{1}{k^2} \ \text{ if } \ \ell \leq k - 1 \ .$$

Hence if T' is an optimum tree, then $\ell = k$ and

$$\sigma(T') = 1 + \frac{1}{k(2k + j)} \ .$$

Thus $\sigma(T')$ is maximum when j is minimum. Consequently, T' is optimal, if, and only if, T contains U and j is minimum. But then T is an optimal Steiner tree in G, and the statement follows. □

Lemma 3.4 implies that if we could solve the Densest Tree problem in polynomial time, we could also solve the Steiner Tree problem in polynomial time. Since the Steiner Tree problem is NP-hard, so is the Steiner Tree problem.

4 Conclusions

In this Thesis we considered several "capacitated" max-density problems. We gave a systematic description of several polynomial time algorithms for the Densest Subgraph problem. We have also formalized a general framework to attack some density problems based on decompositions of feasible solutions, and used it to derive a PTAS for the the Densest Path and the Densest Tree problems; these two problems were shown to be NP-hard. An open question is to develop some generic approach to handle the case when there are weights on the nodes.

References

[1] S. Arora, D. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. *J. Comput. Syst. Sci.*, 58(1):193–210, 1999.

[2] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan. Detecting high log-densities: an $O(n^{1/4})$ approximation for densest k-subgraph. In *STOC*, pages 201–210, 2010.

[3] C. Blum. Revisiting dynamic programming for finding optimal subtrees in trees. *European Journal of Operational Research*, 177:102115, 2007.

[4] C. Blum and M. Ehrgott. Local search algorithms for the k-cardinality tree problem. *Discrete Applied Mathematics*, 128:511540, 2003.

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

[6] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *APPROX*, pages 84–95, 2000.

[7] C. Chekuri and N. Korula. Pruning 2-connected graphs. In *FSTTCS*, 2008.

[8] D. G. Corneil and Y. Perl. Clustering and domination in perfect graphs. *Discrete Applied Mathematics*, pages 27–39, 1984.

[9] J.-P. Crouzeix and J. A. Ferland. Algorithms for generalized fractional programming. *Math. Program.*, 52(2):191–207, 1991.

[10] G. Even. Recursive greedy methods. In T. F. Gonzales, editor, *Handbook of Approximation Algorithms and Metaheuristics*. Chapman & Hall/CRC, 2007.

[11] U. Feige, G. Kortsarz, and D. Peleg. The dense k-subgraph problem. *Algorithmica*, 29:410–421, 2001.

[12] U. Feige and M. Seltser. On the densest k-subgraph problem. Technical Report CS97-16, Weizmann Institute, 1997.

[13] M. Feldman and Z. Nutov. Greedy methods in approximation algorithms. Manuscript, 2008.

[14] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.*, 18:30–55, February 1989.

[15] D. Gibson, J. Kleinberg, and P. Raghavan. Inferring web communities from link topology. In *HYPERTEXT*, pages 225–234, 1998.

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

[17] R. Hassin, S. Rubinstein, and A. Tamir. Approximation algorithms for maximum dispersion. *Operations Research Letters*, 21(3):133–137, 1997.

[18] D. J. Huang and A. B. Kahng. When clusters meet partitions: A new density objective for circuit decomposition. In *ED&TC*, pages 60–64, 1995.

[19] J. M. Keil and T. B. Brecht. The complexity of clustering in planar graphs. *J. Combinatorial Mathematics and Combinatorial Computing*, 9:155–159, 1991.

[20] S. Khot. Ruling out ptas for graph min-bisection, dense k-subgraph, and bipartite clique. *SIAM J. Comput*, 36(4):1025–1071, 2006.

[21] M. Liazi, I. Milis, and V. Zissimopoulos. A constant approximation algorithm for the densest k-subgraph problem on chordal graphs. *Inf. Process. Lett.*, 108(1):29–32, 2008.

[22] F. Maffioli. Finding a best subtree of a tree. Technical Report 91.041, Politecnico di Milano, Dipartimento di Elettronica, Italy, 1991.

[23] B. F. Miles and V. V. Phoha. The bipartite clique: a topological paradigm for wwweb user search customization. In *ACM-SE*, pages 90–95, 2005.

[24] J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. In *Operations Research*, pages 377–387, 1988.

[25] A. Schrijver. *Combinatorial Optimization, Polyhedra and Efficiency.* Springer-Verlag Berlin, Heidelberg New York, 2004.