Report Number: WUCSE-2006-26

2006-01-01

The Remote-Clique Problem Revisited

Benjamin E. Birnbaum

Given a positive integer k and a complete graph with non-negative edge weights that satisfy the triangle inequality, the remote-clique problem is to find a subset of k vertices having a maximum-weight induced subgraph. A greedy algorithm for the problem has been shown to have an approximation ratio of 4, but this analysis was not shown to be tight. In this thesis, we present an algorithm called d-Greedy Augment that generalizes this greedy algorithm (they are equivalent when d = 1). We use the technique of factor-revealing linear programs to prove that d-Greedy Augment, which has a running time... Read complete abstract on page 2.

The Remote-Clique Problem Revisited

Benjamin E. Birnbaum

Complete Abstract:

Given a positive integer k and a complete graph with non-negative edge weights that satisfy the triangle inequality, the remote-clique problem is to find a subset of k vertices having a maximum-weight induced subgraph. A greedy algorithm for the problem has been shown to have an approximation ratio of 4, but this analysis was not shown to be tight. In this thesis, we present an algorithm called d-Greedy Augment that generalizes this greedy algorithm (they are equivalent when d = 1). We use the technique of factor-revealing linear programs to prove that d-Greedy Augment, which has a running time of O(kdnd), achieves an approximation ratio of (2k ? 2)/(k + d ? 2). Thus, when d = 1, d-Greedy Augment achieves an approximation ratio of 2 and runs in time O(kn), making it the fastest known 2-approximation for the remote-clique problem. Beyond proving this worst-case result, we also examine the behavior of d-Greedy Augment in practice. First, we provide some theoretical results regarding the expected case performance of d-Greedy Augment on random graphs, and second, we describe data from some experiments that test the performance of d-Greedy Augment and related heuristics.

2006-26

The Remote-Clique Problem Revisited

Authors: Benjamin E. Birnbaum

Corresponding Author: kjg@cse.wustl.edu

Abstract: Given a positive integer k and a complete graph with non-negative edge weights that satisfy the triangle inequality, the remote-clique problem is to find a subset of k vertices having a maximum-weight induced subgraph. A greedy algorithm for the problem has been shown to have an approximation ratio of 4, but this analysis was not shown to be tight. In this thesis, we present an algorithm called d-Greedy Augment that generalizes this greedy algorithm (they are equivalent when d = 1). We use the technique of factor-revealing linear programs to prove that d-Greedy Augment, which has a running time of O(kdnd), achieves an approximation ratio of (2k ? 2)/(k + d ? 2). Thus, when d = 1, d-Greedy Augment achieves an approximation ratio of 2 and runs in time O(kn), making it the fastest known 2-approximation for the remote-clique problem. Beyond proving this worst-case result, we also examine the behavior of d-Greedy Augment in practice. First, we provide some theoretical results regarding the expected case performance of d-Greedy Augment on random graphs, and second, we describe data from some experiments that test the performance of d-Greedy Augment and related heuristics.

Type of Report: Other

The Remote-Clique Problem Revisited

Undergraduate Honors Thesis

Benjamin E. Birnbaum

Advisors:
Jonathan S. Turner
Kenneth J. Goldman

May, 2006

Department of Computer Science and Engineering
School of Engineering and Applied Science
Washington University in St. Louis

ABSTRACT

Given a positive integer k and a complete graph with non-negative edge weights that satisfy the triangle inequality, the *remote-clique* problem is to find a subset of k vertices having a maximum-weight induced subgraph. A greedy algorithm for the problem has been shown to have an approximation ratio of 4, but this analysis was not shown to be tight. In this thesis, we present an algorithm called d-Greedy Augment that generalizes this greedy algorithm (they are equivalent when $d = 1$). We use the technique of *factor-revealing linear programs* to prove that d-Greedy Augment, which has a running time of $O(kdn^d)$, achieves an approximation ratio of $(2k - 2)/(k + d - 2)$. Thus, when $d = 1$, d-Greedy Augment achieves an approximation ratio of 2 and runs in time $O(kn)$, making it the fastest known 2-approximation for the remote-clique problem. Beyond proving this worst-case result, we also examine the behavior of d-Greedy Augment in practice. First, we provide some theoretical results regarding the expected case performance of d-Greedy Augment on random graphs, and second, we describe data from some experiments that test the performance of d-Greedy Augment and related heuristics.

ACKNOWLEDGEMENTS

I wish to thank both of my advisors, Jon Turner and Ken Goldman, for many hours of helpful guidance and support. This work was supported in part by the National Science Foundation under CISE Educational Innovation grant 0305954.

1. INTRODUCTION

As computing system reliability becomes increasingly important, techniques are being developed to ensure that essential services can survive in spite of a limited number of arbitrary faults, including communication failures, crash failures, software errors, and malicious attacks. These techniques fundamentally require replication of data and processing so that when failures occur, other replicas can continue to correctly execute operations to completion. As an example, the CLBFT algorithm [2], uses a group of $3f + 1$ replicated data servers to guarantee correct performance when up to f of the replica servers exhibit faulty behavior. However, such algorithms are only beneficial if the faults are *independent*. Otherwise, a single failure type (such as a power outage affecting all servers in a region) could induce faults in multiple server replicas, exceeding the fault-tolerance bound f. Therefore, provided that all replicas implement the same specification, it is desirable that servers of a given replica group are as diverse as possible with repsect to properties such as system architecture, software, physical location, and the network administrator. Motivated by such concerns, we investigate the problem of choosing a set of $3f + 1$ replicas from a large group of potential servers when it is important to choose a set of replicas that are as diverse as possible.

We can model this situation as a graph problem. For each potential server, create a vector of attributes representing its properties (architecture, software, location, etc.) and define the distance between two servers to be the *Hamming distance* between the vectors (i.e. the number of times corresponding attributes differ.) For each server, create a vertex in a complete graph. Set the edge weight between two vertices to be the distance between their corresponding servers. Finding a subset of servers that is most diverse now becomes the problem of finding a subset of $3f + 1$ vertices having an induced subgraph with maximum average edge weight (or equivalently maximum total edge weight).

This example provides one of several motivations for a general class of graph problems called *dispersion* problems, which involve finding subsets of vertices that are in some way as distant from each other as possible. (See [3] for other motivating examples.) Maximizing the average weight of the induced subgraph is the measure of dispersion addressed here, but it is possible to measure dispersion in a number of other ways, including the minimum weight edge and the minimum weight spanning tree of the induced subgraph. As in [3], we call the problem of maximizing the average weight the *remote-clique* problem, but it has also been called *maxisum dispersion* [7] and *max-avg facility dispersion* [11].

We define the problem more formally as follows. Let $G = (V, E)$ be a complete graph with

the weight for edge $(v_1, v_2) \in E$ given by $w(v_1, v_2)$. (Define $w(v, v) = 0$ for all $v \in V$.) The edge weights are nonnegative and satisfy the triangle inequality: for all $v_1, v_2, v_3 \in V$, $w(v_1, v_2) + w(v_2, v_3) \geq w(v_1, v_3)$. [1] For a given integer parameter k, such that $1 \leq k \leq |V|$, the *remote-clique problem* is to find a subset $V' \subseteq V$ such that $|V'| = k$ and the average edge weight in V', $2/(k(k-1)) \cdot \sum_{\{v_1,v_2\} \in E \,:\, v_1,v_2 \in V'} w(v_1, v_2)$, is maximized. This problem can be shown to be NP-hard by an easy reduction from CLIQUE.

It has been shown that a simple greedy algorithm with running time $O(n^2)$ achieves an approximation ratio of 4 (i.e., it always finds a solution that has an average edge weight no less than $\frac{1}{4}$ times optimal) [11]. An example is provided in which the optimal solution weighs twice as much as the algorithm's solution, but the question of whether a tighter bound for the algorithm can be proved remained open. In another paper, it is proved that a more complicated algorithm with running time $O(n^2 + k^2 \log k)$ achieves an approximation ratio of 2 [5]. However, a tight approximation ratio for the simple greedy algorithm has never been proved.

In this thesis, we provide an algorithm parameterized by an integer d called d-GREEDY AUGMENT. When $d = 1$, the algorithm is nearly the same as the algorithm analyzed in [11], and when $d = k$, the algorithm amounts to examining all subsets of k vertices and choosing the one with maximum edge weight. (Clearly, this will return an optimal solution, but it does not run in polynomal time if k is not a constant.) We will show that d-GREEDY AUGMENT has an approximation ratio of $(2k-2)/(k+d-2)$ and has a running time of $O(kdn^d)$. Therefore if $d = 1$, our algorithm guarantees an approximation ratio of 2 and has a running time of $O(kn)$. This algorithm, then, is the fastest known (and easiest to implement) 2-approximation for the remote-clique problem. Our proof of the approximation ratio, which uses the technique of *factor-revealing linear programs* [6, 9], also answers an open question from [11] by showing that the greedy algorithm does indeed obtain an approximation ratio of 2.

We also provide some results regarding the expected characteristics of remote-clique problem instances and the performance of d-GREEDY AUGMENT. For these probabilistic analyses, we make the simplifying assumption that the graph is unweighted (or equivalently is weighted in which the edge weight is always 0 or 1). In this context, the remote-clique problem becomes the problem of finding a group of k vertices that has maximum average degree. An algorithm achieving an approximation ratio slightly better than $O(n^{1/3})$ is provided in [4], in which the problem was called the Dense k-Subgraph Problem. No matching hardness results exist for this problem, but it is conjectured that there is some universal constant ϵ such that it is NP-hard to approximate the problem to within a factor of $O(n^\epsilon)$. To the best of our knowledge, however, there has not been an investigation into the average case performance of algorithms for this problem.

Beyond these theoretical results, we also analyze the performance of d-GREEDY AUGMENT and related heuristics experimentally. We provide evidence showing that d-GREEDY AUG-

[1] It can easily be shown that the Hamming distance defined earlier does indeed satisfy the triangle inequality.

MENT seems to perform much better in practice than its worst-case analysis might suggest. We also show that a simple modification of d-GREEDY AUGMENT called ALL-GREEDY AUGMENT seems to perform even better.

This thesis is organized as follows. The remainder of this section establishes some notation and combinatorial identities that will be used later in the thesis. Section 2 proves the approximation ratio of $(2k-2)/(k+d-2)$ for d-GREEDY AUGMENT and shows that this ratio is a tight bound on the worst case performance of the algorithm. In Section 3, we establish some results regarding the expected behavior of d-GREEDY AUGMENT on random graph instances. In Section 4, we report the results from our experiments comparing the performance of d-GREEDY AUGMENT and related heuristics on randomly generated problem instances. We conclude in Section 5.

1.1 Notation

We define the following notation that will be used throughout this thesis:

- For any natural number n, let $[n] = \{1, \ldots, n\}$.
- For any integers n and j such that $n \geq 1$ and $j \leq n$, let

$$\binom{n}{j} = \begin{cases} \frac{n!}{j!(n-j)!} & \text{if } j \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

  There will be several times in this thesis when we will write $\binom{n}{j}$ when j could be less than 0, so it is important to keep in mind the full definition of $\binom{n}{j}$.

- For any set S and nonnegative integer j, let $\binom{S}{j} = \{S' \subseteq S \ : \ |S'| = j\}$. If $j \leq 0$ or $j > |S|$, then $\binom{S}{j} = \emptyset$.

1.2 Combinatorial Identities

In this section, we state two combinatorial identities that will be used later in this thesis. First, the following fact about summations will be used a number of times in Section 2.

Lemma 1. *Let n be a positive integer and let i and j be two nonnegative integers such that $j \leq i \leq n$. Then if f is some fuction defined on the domain $\binom{[n]}{j}$, the following identity holds.*

$$\sum_{I \in \binom{[n]}{i}} \sum_{J \in \binom{I}{j}} f(J) = \binom{n-j}{i-j} \sum_{J \in \binom{[n]}{j}} f(J)$$

Proof. To prove this, we count the number of times that a given $J \in \binom{[n]}{j}$ appears in the summation on the left. Each J appears once for each $I \in \binom{[n]}{i}$ containing J. The number of such sets I is $\binom{n-j}{i-j}$, since for a given J, we can choose $i-j$ elements from the $n-j$ elements in $[n]-J$ to form such an I. □

The following fact will also be used in Section 2.

Lemma 2. *For all integers j, d, and s, such that $0 \leq j \leq d \leq s$,*

$$\binom{s}{d}\binom{d}{j} = \binom{s}{j}\binom{s-j}{d-j}$$

Proof. By definition, we have

$$\binom{s}{d}\binom{d}{j} = \frac{s!}{d!(s-d)!}\frac{d!}{j!(d-j)!} = \frac{s!}{j!(d-j)!(s-d)!}$$

and

$$\binom{s}{j}\binom{s-j}{d-j} = \frac{s!}{j!(s-j)!}\frac{(s-j)!}{(d-j)!(s-d)!} = \frac{s!}{j!(d-j)!(s-d)!}$$

Hence

$$\binom{s}{d}\binom{d}{j} = \binom{s}{j}\binom{s-j}{d-j}$$

□

2. WORST-CASE ANALYSIS OF d-GREEDY AUGMENT

In this section, we describe d-GREEDY AUGMENT and analyze its running time and worst-case performance. d-GREEDY AUGMENT maintains a set of vertices T that starts empty. At each step in the algorithm, it augments T with the set of d vertices that will add the most weight to T. When $|T| = k$, d-GREEDY AUGMENT returns the set T. (Throughout this paper, we assume for simplicity that d divides k evenly.) To be more precise, we define the following notation. For any subset of vertices $V' \subseteq V$ that is disjoint from T, let

$$\text{aug}_T(V') = \sum_{v' \in V'} \sum_{v \in T} w(v', v) + \sum_{\{v'_1, v'_2\} \in \binom{V'}{2}} w(v'_1, v'_2)$$

In other words, $\text{aug}_T(V')$ is the amount of edge weight that the vertices in V' add to T if T is augmented by V'. At each step in the algorithm, d-GREEDY AUGMENT chooses a set of d vertices V' that maximizes $\text{aug}_T(V')$ and adds them to T. For the first step, this means that d-GREEDY AUGMENT chooses a group of d vertices with the heaviest edge weights.[1]

An implementation of d-GREEDY AUGMENT is listed below as Algorithm 1. The running time is dominated by the outer while loop, which runs k/d times. Each iteration of the while loop has $O(n^d)$ iterations through the subsets V', and each of these iterations takes $O(d^2)$ time to compute the sum to add to $aug(V')$. Therefore, the overall running time of this algorithm is $O(kdn^d)$.

Because the running time of d-GREEDY AUGMENT is exponential in d, it only runs in polynomial time for constant values of d. Furthermore, the remote-clique problem is only NP-hard for non-constant values of k (since otherwise we could just examine every subset of k vertices and choose the maximum weight subset). Therefore, increasing the value of d does not affect the approximation ratio asymptotically. However, we include the analysis for all values of d both for completeness and because it could have some practical benefit for small values of k. For example, if $k = 4$, then the naive exact algorithm would take quartic time, whereas if we were willing to spend quadratic time, we could run d-GREEDY AUGMENT for $d = 2$ to guarantee finding a solution at least $\frac{2}{3}$ the value of the optimal solution.

To prove our approximation ratio of $(2k - 2)/(k + d - 2)$, we use the technique of *factor-revealing linear programs* [6, 9], which is a simple generalization of a method often used to

[1] Note that if $d = 1$, then during the first step $\text{aug}_T(V') = 0$ for all $V' \subseteq V$ such that $|V'| = d$. Thus for $d = 1$, d-GREEDY AUGMENT just starts with an arbitrary vertex. This is in fact the only difference between d-GREEDY AUGMENT for $d = 1$ and the algorithm analyzed in [11]; the algorithm in [11] initializes T to two vertices that are endpoints of a maximum weight edge.

Algorithm 1 d-Greedy Augment

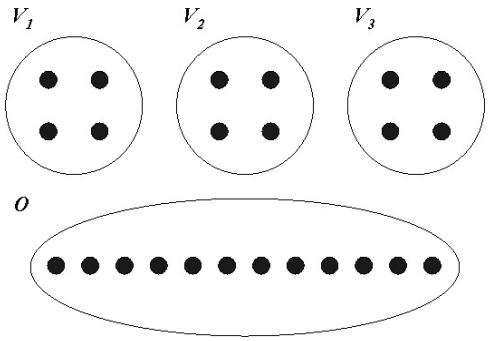
for all $V' \subseteq V$ such that $|V'| = d$ **do**
  $aug(V') \leftarrow \sum_{\{v'_1, v'_2\} \in \binom{V'}{2}} w(v'_1, v'_2)$
end for
$T \leftarrow \emptyset$
while $|T| < k$ **do**
  $MaxWeight \leftarrow -\infty$
  for all $V' \subseteq V - T$ such that $|V'| = d$ **do**
    if $aug(V') > MaxWeight$ **then**
      $MaxWeightSet \leftarrow V'$
      $MaxWeight \leftarrow aug(V')$
    end if
  end for
  $T \leftarrow T \cup MaxWeightSet$
  for all $V' \subseteq V - T$ such that $|V'| = d$ **do**
    $aug(V') \leftarrow aug(V') + \sum_{v \in MaxWeightSet} \sum_{v' \in V'} w(v, v')$
  end for
end while
return T

provide bounds for approximation algorithms. Consider a maximization (resp., minimization) problem P. A typical analysis of an approximation algorithm ALG for P proceeds by using the behavior of ALG and the structure of P to generate a number of inequalities. These inequalities are then combined to provide a bound on the value of the solution obtained by ALG to that of an optimal solution. Often, this can be done algebraically, but not always. A more general way of obtaining a bound is to view the process as an optimization problem Q in its own right, in which an adversary tries to minimize (resp., maximize) the value of ALG's solution to P subject to the constraints given by the generated inequalities. The optimal solution to Q is then a bound on the performance of ALG. If Q can be formulated as a linear program, then this is a *factor-revealing LP*, which can then be solved using duality. The simplicity of this technique makes it applicable to many problems, but in most cases it does not seem to be the easiest way to provide a bound. However, there are some algorithms, including the one in [6] and the greedy algorithm examined here, in which it is the only known technique to provide a tight bound.

Before we begin with the proof that d-Greedy Augment obtains an approximation ratio of $(2k-2)/(k+d-2)$, we observe with the following theorem that we cannot hope to prove a better approximation ratio for this algorithm.

Theorem 3. *There exist infinitely many remote-clique problem instances in which the ratio of the average edge weight in an optimal solution to the average edge weight in the solution returned by* d-Greedy Augment *is* $(2k-2)/(k+d-2)$.

Fig. 2.1: An example (for $k = 12$ and $d = 4$) showing that the approximation ratio of $(2k-2)/(k+d-2)$ is a tight bound on the worst-case performance of d-Greedy Augment. Edges that are contained within the same circle have weight 2, and all other edges have weight 1.



Proof. Consider the following problem instance $(G = (V, E), k)$, in which $|V| = 2k$ and V is partitioned into k/d subsets of d vertices called $V_1, \ldots, V_{k/d}$ and one group of k vertices called O. The edge weights are determined as follows:

$$w(v_1, v_2) = \begin{cases} 2 & \text{if } v_1, v_2 \in V_i \text{ for some i or } v_1, v_2 \in O \\ 1 & \text{otherwise} \end{cases}$$

This construction is illustrated in Figure 2.1. It is clear that the edges in this problem instance satisfy the triangle inequality, since every edge has weight either 1 or 2. Also, it is clear that d-Greedy Augment could (if ties were broken badly) return the solution $T = \bigcup_i V_i$, whereas the optimal solution is O. If this happens, then the total edge weight in T is

$$\frac{k}{d}\binom{d}{2} \cdot 2 + \left(\binom{k}{2} - \frac{k}{d}\binom{d}{2}\right) \cdot 1 = \frac{1}{2}k(k+d-2)$$

and since the total edge weight in O is $2\binom{k}{2}$, the ratio of the performance of an optimal algorithm to d-Greedy Augment is $(2k-2)/(k+d-2)$. To see that there are an infinite number of such graphs, note that without affecting the relative performance of d-Greedy Augment, we can add arbitrarly many vertices to this construction in which every edge incident to these new vertices has weight 1. □

We now continue with a proof of our approximation ratio. For an instance of the remote-clique problem, let OPT be the average edge weight in an optimal solution. To prove that d-Greedy Augment achieves an approximation ratio of $(2k-2)/(k+d-2)$, we will prove that at each augmenting step, there exists a group of d vertices V' in which $\text{aug}_T(V')$ is sufficiently high.

Lemma 4. *Before each augmenting step in the algorithm, there exists a group of d vertices $V' \subseteq V$ such that V' is disjoint from T and* $\text{aug}_T(V') \geq \frac{1}{2}d(|T| + d - 1)OPT$.

Proof. We defer the proof of the lemma when $|T| > 0$ to the remainder of this section. When $|T| = 0$, then the statement of the lemma is that there exists at least one group of vertices $V' \subseteq V$ of size d and total weight at least $\binom{d}{2}OPT$. Let S be the set vertices in an optimal solution. We prove that such a group of vertices V' must be found as a subset of S. By the optimality of S,

$$\sum_{\{v_1,v_2\}\in\binom{S}{2}} w(v_1, v_2) = \binom{k}{2}OPT = \frac{\binom{k}{d}\binom{d}{2}}{\binom{k-2}{d-2}}OPT$$

where the second equality follows by Lemma 2. Thus,

$$\binom{k-2}{d-2}\sum_{\{v_1,v_2\}\in\binom{S}{2}} w(v_1, v_2) = \binom{k}{d}\binom{d}{2}OPT$$

But by Lemma 1, we have

$$\sum_{V'\in\binom{S}{d}}\sum_{\{v_1,v_2\}\in\binom{V'}{2}} w(v_1, v_2) = \binom{k}{d}\binom{d}{2}OPT$$

which implies the lemma for $|T| = 0$. □

With this fact, it is straightforward to prove that d-Greedy Augment achieves an approximation ratio of $(2k-2)/(k+d-2)$.

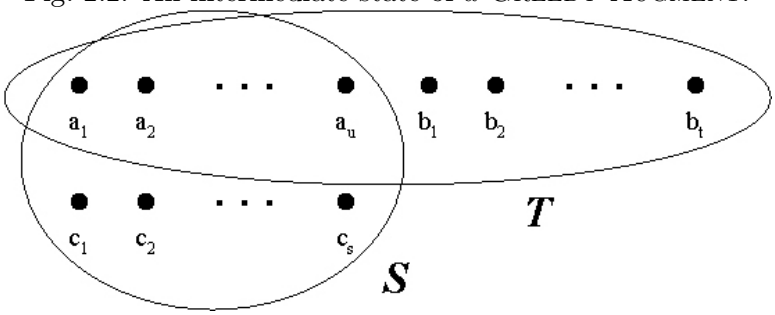
Theorem 5. *The average weight of the edges in the solution returned by* d-Greedy Augment *is at least* $(k+d-2)/(2k-2)\cdot OPT$.

Proof. Since d-Greedy Augment adds d vertices to T during each augmenting step, we have that $|T| = (i-1)d$ before the i^{th} augmenting step. Thus by Lemma 4, there exists a set of d vertices $V' \subseteq V$ that can be added to T such that $\text{aug}_T(V') = \frac{1}{2}d(di-1)OPT$. We know therefore that the weight of the edges added by d-Greedy Augment on the i^{th} augmenting step is at least $\frac{1}{2}d(di-1)OPT$, since d-Greedy Augment chooses the set of d vertices that maximizes $\text{aug}_T(V')$. Thus after j steps, the weight of the edges in T is at least

$$\sum_{i=1}^{j}\frac{1}{2}d(di-1)OPT = \frac{1}{4}dj(dj+d-2)OPT$$

Since the algorithm terminates after k/d steps, the final total weight of the edges in T is at least $\frac{1}{4}k(k+d-2)OPT$. Since there are $k(k-1)/2$ edges in this set of vertices, we conclude that the average weight of the edges in the solution returned by d-Greedy Augment is at least $(k+d-2)/(2k-2)\cdot OPT$. □

Fig. 2.2: An intermediate state of d-Greedy Augment.



To prove Lemma 4, we begin with some notation. Consider an intermediate state of d-Greedy Augment. The set of vertices chosen so far is T, and let S be the set of vertices in an optimal solution. Let $u = |S \cap T|$, $t = |T - S|$, and $s = |S - T|$. Arbitrarily label the vertices in $S \cap T$ as $a_1, a_2, \ldots, a_u$, the vertices in $T - S$ as $b_1, b_2, \ldots, b_t$, and the vertices in $S - T$ as $c_1, c_2, \ldots, c_s$. Note that one of u or t may be equal to zero, but since we are in an intermediate state of d-Greedy Augment, we know that $d \leq |T| \leq |S| - d$, and hence $u + t \geq d$ and $s \geq t + d$. This situation is illustrated in Figure 2.2.

We break the proof of Lemma 4 into five cases based on the values of u and t and state the proof for each case as its own lemma. For each of these cases, we show that a group of d vertices satisfying the condition of Lemma 4 can be found in the set $S - T$. We start with the case when S and T are disjoint, i.e., when $u = 0$ and $t \geq 1$. This case permits a direct analysis without the use of linear programming. It is instructive to analyze this case first, since it will be become more clear why imitating this analysis does not seem to be possible for the other cases. This should help motivate the use of the more general technique of factor-revealing linear programs for the other cases.

Lemma 6. *Lemma 4 holds when $u = 0$ and $t \geq 1$. Specifically, there exists a set of indices $L \in \binom{[s]}{d}$ such that* [2]

$$\sum_{\ell \in L} \sum_{j \in [t]} w(b_j, c_\ell) + \sum_{\{\ell, m\} \in \binom{L}{2}} w(c_\ell, c_m) \geq \frac{1}{2} d(d + t - 1) OPT$$

Proof. The key observation is that because of the triangle inequality, edges adjacent to the high-weight edges in S must also have high weight on average. By the triangle inequality,

$$w(b_j, c_\ell) + w(b_j, c_m) \geq w(c_\ell, c_m) \qquad j \in [t], \{\ell, m\} \in \binom{[s]}{2}$$

[2] Note that if $d = 1$, then the second sum in this inequality is empty. In general, there will be a number of formulas in this section that simplify significantly if $d = 1$. However, we use the general form for compactness, and unless otherwise noted, all statements hold for any $d \geq 1$.

Summing over all j, this becomes

$$\sum_{j\in[t]} w(b_j, c_\ell) + \sum_{j\in[t]} w(b_j, c_m) \geq tw(c_\ell, c_m) \qquad \{\ell, m\} \in \binom{[s]}{2}$$

Now, summing over all $\{\ell, m\}$ yields

$$\sum_{\{\ell,m\}\in\binom{[s]}{2}} \left(\sum_{j\in[t]} w(b_j, c_\ell) + \sum_{j\in[t]} w(b_j, c_m) \right) \geq t \sum_{\{\ell,m\}\in\binom{[s]}{2}} w(c_\ell, c_m) = t\binom{s}{2} OPT \tag{2.1}$$

where the equality follows from the optimality of S. By applying Lemma 1 to the lefthand side of (2.1), we can rewrite it as

$$(s-1)\sum_{\ell\in[s]}\sum_{j\in[t]} w(b_j, c_\ell) \geq t\binom{s}{2} OPT$$

which can be simplified to

$$\sum_{\ell\in[s]}\sum_{j\in[t]} w(b_j, c_\ell) \geq \frac{st}{2} OPT$$

From this fact, along with the optimality of S, it follows that [3]

$$\begin{aligned} \binom{s-1}{d-1}\sum_{\ell\in[s]}\sum_{j\in[t]} w(b_j, c_\ell) + \binom{s-2}{d-2}\sum_{\{\ell,m\}\in\binom{[s]}{2}} w(c_\ell, c_m) &\geq \left(\binom{s-1}{d-1}\frac{st}{2} + \binom{s-2}{d-2}\binom{s}{2} \right) OPT \\ &= \frac{1}{2}\binom{s}{d} d(d+t-1) OPT \end{aligned} \tag{2.2}$$

where the equality is obtained from some simplification using Lemma 2. We can now apply Lemma 1 to the lefthand side of this inequality to obtain

$$\sum_{L\in\binom{[s]}{d}} \left(\sum_{\ell\in L}\sum_{j\in[t]} w(b_j, c_\ell) + \sum_{\{\ell,m\}\in\binom{L}{2}} w(c_\ell, c_m) \right) \geq \frac{1}{2}\binom{s}{d} d(d+t-1) OPT$$

But this implies that there must exist at least one $L \in \binom{[s]}{d}$ such that

$$\sum_{\ell\in L}\sum_{j\in[t]} w(b_j, c_\ell) + \sum_{\{\ell,m\}\in\binom{L}{2}} w(c_\ell, c_m) \geq \frac{1}{2} d(d+t-1) OPT$$

□

[3] Again, note the difference in the formula if $d = 1$. In this case, $\binom{s-2}{d-2} = 0$.

Now that we have proved Lemma 4 for the case when T is disjoint from S, we turn to proving Lemma 4 when some number of vertices in T are also in S. Intuitively, the algorithm should do no worse when T is not disjoint from S. If d-Greedy Augment has already found some of the optimal solution, then that should not hurt its performance. However, we will see that this case seems harder to analyze, and we will need to use factor-revealing linear programs for this case. We start with the most general non-disjoint case that we examine, when $u \geq 2$ and $t \geq 1$. (Our application of the triangle inequality leads to certain boundary cases that arise for other values of u and t, which we handle separately.)

Lemma 7. *Lemma 4 holds when $u \geq 2$ and $t \geq 1$. Specifically, there exists a set of indices $L \in \binom{[s]}{d}$ such that*

$$\sum_{\ell \in L} \left(\sum_{h \in [u]} w(a_h, c_\ell) + \sum_{j \in [t]} w(b_j, c_\ell) \right) + \sum_{\{\ell, m\} \in \binom{L}{2}} w(c_\ell, c_m) \geq \frac{1}{2} d(d + t + u - 1) OPT$$

Proof. It is sufficient to show that

$$\binom{s-1}{d-1} \sum_{\ell \in [s]} \left(\sum_{h \in [u]} w(a_h, c_\ell) + \sum_{j \in [t]} w(b_j, c_\ell) \right) + \binom{s-2}{d-2} \sum_{\{\ell, m\} \in \binom{[s]}{2}} w(c_\ell, c_m) \geq \frac{1}{2} \binom{s}{d} d(d+t+u-1) OPT \tag{2.3}$$

since we can then proceed to prove this lemma as we proved Lemma 6 from inequality (2.2). To prove inequality (2.3), we proceed as we did in Lemma 6; we use the triangle inequality and the optimality of S. By the triangle inequality, we have

$$w(a_h, c_\ell) + w(a_h, c_m) - w(c_\ell, c_m) \geq 0 \qquad h \in [u], \{\ell, m\} \in \binom{[s]}{2} \tag{2.4}$$

$$w(b_j, c_\ell) + w(b_j, c_m) - w(c_\ell, c_m) \geq 0 \qquad j \in [t], \{\ell, m\} \in \binom{[s]}{2} \tag{2.5}$$

$$w(a_h, c_\ell) + w(a_i, c_\ell) - w(a_h, a_i) \geq 0 \qquad \{h, i\} \in \binom{[u]}{2}, \ell \in [s] \tag{2.6}$$

By the optimality of S, we have

$$\sum_{\{h,i\} \in \binom{[u]}{2}} w(a_h, a_i) + \sum_{\{\ell, m\} \in \binom{[s]}{2}} w(c_\ell, c_m) + \sum_{h \in [u]} \sum_{\ell \in [s]} w(a_h, c_\ell) \geq \binom{s+u}{2} OPT \tag{2.7}$$

At this point in the proof of Lemma 6, it was possible to combine the inequalities expressing the triangle inequality and the optimality of S to yield (2.2) and finish the proof. In this lemma, however, the inequalities have a much more complicated form because of the overlap of S and T, and it does not seem possible to combine them directly to yield (2.3). To prove

(2.3), we instead consider an adversary trying to minimize

$$\binom{s-1}{d-1}\sum_{\ell\in[s]}\left(\sum_{h\in[u]}w(a_h,c_\ell)+\sum_{j\in[t]}w(b_j,c_\ell)\right)+\binom{s-2}{d-2}\sum_{\{\ell,m\}\in\binom{[s]}{2}}w(c_\ell,c_m)$$

subject to the constraints given by (2.4), (2.5), (2.6), and (2.7). If we can show that the optimal value of this factor-revealing linear program (where the variables are the weights of the edges) is at least $\frac{1}{2}\binom{s}{d}d(d+t+u-1)OPT$, then we will have proven (2.3). Since the value of any feasible dual solution is a lower bound for the optimal value of the primal, we can prove (2.3) by finding a feasible dual solution with value $\frac{1}{2}\binom{s}{d}d(d+t+u-1)OPT$. The dual linear program is

maximize

$$\binom{s+u}{2}OPT\cdot z$$

subject to

$$\begin{aligned}
-\sum_{\ell\in[s]}y_{\{h,i\},\ell}+z&\le 0 && \{h,i\}\in\binom{[u]}{2}\\
-\sum_{h\in[u]}w_{h,\{\ell,m\}}-\sum_{j\in[t]}x_{j,\{\ell,m\}}+z&\le\binom{s-2}{d-2} && \{\ell,m\}\in\binom{[s]}{2}\\
\sum_{m\in[s]-\{\ell\}}w_{h,\{\ell,m\}}+\sum_{i\in[u]-\{h\}}y_{\{h,i\},\ell}+z&\le\binom{s-1}{d-1} && h\in[u],\ell\in[s]\\
\sum_{m\in[s]-\{\ell\}}x_{j,\{\ell,m\}}&\le\binom{s-1}{d-1} && j\in[t],\ell\in[s]
\end{aligned}\tag{2.8}$$

where $w_{h,\{\ell,m\}}$ corresponds to (2.4), $x_{j,\{\ell,m\}}$ corresponds to (2.5), $y_{\{h,i\},\ell}$ corresponds to (2.6), and z corresponds to (2.7). It can be easily verified that the following dual solution is feasible.

$$\begin{aligned}
w'_{h,\{\ell,m\}}&=\frac{s-d-t+1}{(u+s)(s-1)}\binom{s-1}{d-1} && h\in[u],\{\ell,m\}\in\binom{[s]}{2}\\
x'_{j,\{\ell,m\}}&=\frac{1}{s-1}\binom{s-1}{d-1} && j\in[t],\{\ell,m\}\in\binom{[s]}{2}\\
y'_{\{h,i\},\ell}&=\frac{d+t+u-1}{(u+s)(u+s-1)}\binom{s-1}{d-1} && \{h,i\}\in\binom{[u]}{2},\ell\in[s]\\
z'&=\frac{d(d+t+u-1)}{(u+s)(u+s-1)}\binom{s}{d}
\end{aligned}$$

The only constraint that is not trivial to verify is (2.8), but some straightforward manipulation shows that if $d = 1$, then the lefthand side is equal to

$$-\frac{su(u+t)}{(u+s)(u+s-1)(s-1)}$$

which is no greater than 0 since $s \geq 2$ when $t \geq 1$. Similarly, if $d > 1$, then the lefthand side can be written as

$$\left(1 - \frac{su(d+t+u-1)}{(u+s)(u+s-1)(d-1)}\right)\binom{s-2}{d-2}$$

which is clearly no greater than $\binom{s-2}{d-2}$. We conclude the proof by noting that the value of this dual solution is

$$\binom{s+u}{2}\binom{s}{d}\frac{d(d+t+u-1)}{(u+s)(u+s-1)}OPT = \frac{1}{2}\binom{s}{d}d(d+t+u-1)OPT$$

which implies that the optimal value of the primal is no less than $\frac{1}{2}\binom{s}{d}d(d+t+u-1)OPT$ and hence implies inequality (2.3), thus proving the lemma. □

We have now proved the most general (and hardest) case of Lemma 4. It remains only to prove three boundary cases. The next case we consider is when $u = 1$ and $t \geq 1$. Using factor-revealing linear programming seems to be necessary for this case as well, but we need to consider it seperately because here we do not have constraint (2.6) (or dual variables of the form $y_{\{h,i\},\ell}$). Because the proof is so similar to the proof of Lemma 7, we merely state the primal and dual factor-revealing linear program and give a feasible solution to the dual that provides a sufficient lower bound for the optimal value of the primal.

Lemma 8. *Lemma 4 holds when $u = 1$ and $t \geq 1$. Specifically, there exists a set of indices $L \in \binom{[s]}{d}$ such that*

$$\sum_{\ell \in L}\left(w(a_1, c_\ell) + \sum_{j \in [t]} w(b_j, c_\ell)\right) + \sum_{\{\ell,m\} \in \binom{L}{2}} w(c_\ell, c_m) \geq \frac{1}{2}d(d+t)OPT$$

Proof. As before, it is sufficient to prove that

$$\binom{s-1}{d-1}\sum_{\ell \in [s]}\left(w(a_1, c_\ell) + \sum_{j \in [t]} w(b_j, c_\ell)\right) + \binom{s-2}{d-2}\sum_{\{\ell,m\} \in \binom{[s]}{2}} w(c_\ell, c_m) \geq \frac{1}{2}\binom{s}{d}d(d+t)OPT$$

Again, we prove this by lower-bounding the optimal solution to the following linear program, in which the constraints are again derived by the triangle inequality and the optimality of S.

minimize

$$\binom{s-1}{d-1}\sum_{\ell\in[s]}\left(w(a_1,c_\ell)+\sum_{j\in[t]}w(b_j,c_\ell)\right)+\binom{s-2}{d-2}\sum_{\{\ell,m\}\in\binom{[s]}{2}}w(c_\ell,c_m)$$

subject to

$$\begin{aligned}
w(a_1,c_\ell)+w(a_1,c_m)-w(c_\ell,c_m)&\geq 0 && \{\ell,m\}\in\binom{[s]}{2}\\
w(b_j,c_\ell)+w(b_j,c_m)-w(c_\ell,c_m)&\geq 0 && j\in[t],\{\ell,m\}\in\binom{[s]}{2}\\
\sum_{\{\ell,m\}\in\binom{[s]}{2}}w(c_\ell,c_m)+\sum_{\ell\in[s]}w(a_1,c_\ell)&\geq\binom{s+1}{2}OPT
\end{aligned}$$

The dual of this linear program is

maximize

$$\binom{s+1}{2}OPT\cdot z$$

subject to

$$\begin{aligned}
-x_{\{\ell,m\}}-\sum_{j\in[t]}y_{j,\{\ell,m\}}+z&\leq\binom{s-2}{d-2} && \{\ell,m\}\in\binom{[s]}{2}\\
\sum_{m\in[s]-\{\ell\}}x_{\{\ell,m\}}+z&\leq\binom{s-1}{d-1} && \ell\in[s]\\
\sum_{m\in[s]-\{\ell\}}y_{j,\{\ell,m\}}&\leq\binom{s-1}{d-1} && j\in[t],\ell\in[s]
\end{aligned}$$

We conclude the proof of this lemma by providing the following dual solution, which can easily be shown to be feasible and to have the value $\frac{1}{2}\binom{s}{d}d(d+t)OPT$.

$$\begin{aligned}
x'_{\{\ell,m\}}&=\frac{s-d-t+1}{(s+1)(s-1)}\binom{s-1}{d-1} && \{\ell,m\}\in\binom{[s]}{2}\\
y'_{j,\{\ell,m\}}&=\frac{1}{s-1}\binom{s-1}{d-1} && j\in[t],\{\ell,m\}\in\binom{[s]}{2}\\
z'&=\frac{d(d+t)}{s(s+1)}\binom{s}{d}
\end{aligned}$$

□

The next case we consider is when $u \geq 2$ and $t = 0$. Although the factor-revealing LP is slightly different, the proof for this case is very similar to the proofs of Lemmas 7 and 8.

Lemma 9. *Lemma 4 holds when $u \geq 2$ and $t = 0$. Specifically, there exists a set of indices $L \in \binom{[s]}{d}$ such that*

$$\sum_{\ell \in L} \sum_{h \in [u]} w(a_h, c_\ell) + \sum_{\{\ell,m\} \in \binom{L}{2}} w(c_\ell, c_m) \geq \frac{1}{2} d(d + u - 1) OPT$$

Proof. Again, it is sufficient to prove that

$$\binom{s-1}{d-1} \sum_{\ell \in [s]} \sum_{h \in [u]} w(a_h, c_\ell) + \binom{s-2}{d-2} \sum_{\{\ell,m\} \in \binom{[s]}{2}} w(c_\ell, c_m) \geq \frac{1}{2} \binom{s}{d} d(d + u - 1) OPT$$

which we do by finding a lower bound to the optimal solution of the following linear program.

minimize

$$\binom{s-1}{d-1} \sum_{\ell \in [s]} \sum_{h \in [u]} w(a_h, c_\ell) + \binom{s-2}{d-2} \sum_{\{\ell,m\} \in \binom{[s]}{2}} w(c_\ell, c_m)$$

subject to

$$w(a_h, c_\ell) + w(a_h, c_m) - w(c_\ell, c_m) \geq 0 \qquad h \in [u], \{\ell, m\} \in \binom{[s]}{2}$$

$$w(a_h, c_\ell) + w(a_i, c_\ell) - w(a_h, a_i) \geq 0 \qquad \{h, i\} \in \binom{[u]}{2}, \ell \in [s]$$

$$\sum_{\{h,i\} \in \binom{[u]}{2}} w(a_h, a_i) + \sum_{\{\ell,m\} \in \binom{[s]}{2}} w(c_\ell, c_m) + \sum_{h \in [u]} \sum_{\ell \in [s]} w(a_h, c_\ell) \geq \binom{s+u}{2} OPT$$

The dual of this linear program is

maximize

$$\binom{s+u}{2} OPT \cdot z$$

subject to

$$
\begin{aligned}
-\sum_{\ell\in[s]} y_{\{h,i\},\ell} + z \le 0 \qquad & \{h,i\} \in \binom{[u]}{2} \\
-\sum_{h\in[u]} x_{h,\{\ell,m\}} + z \le \binom{s-2}{d-2} \qquad & \{\ell,m\} \in \binom{[s]}{2} \\
\sum_{m\in[s]-\{\ell\}} x_{h,\{\ell,m\}} + \sum_{i\in[u]-\{h\}} y_{\{h,i\},\ell} + z \le \binom{s-1}{d-1} \qquad & h \in [u], \ell \in [s]
\end{aligned}
$$

The following dual solution is feasible and has value $\frac{1}{2}\binom{s}{d}d(d+u-1)OPT$, thus concluding the proof of the lemma.

$$
\begin{aligned}
x'_{h,\{\ell,m\}} &= \frac{s-d+1}{(u+s)(s-1)}\binom{s-1}{d-1} && h \in [u], \{\ell,m\} \in \binom{[s]}{2} \\
y'_{\{h,i\},\ell} &= \frac{d+u-1}{(u+s)(u+s-1)}\binom{s-1}{d-1} && \{h,i\} \in \binom{[u]}{2}, \ell \in [s] \\
z' &= \frac{d(d+u-1)}{(u+s)(u+s-1)}\binom{s}{d}
\end{aligned}
$$

□

The final case yet to be covered is when $u = 1$ and $t = 0$. If this is true, then it must be the case that $d = 1$, since $u + t \ge d$. The proof of this case follows from a simple contradiction argument.

Lemma 10. *Lemma 4 holds when $u = 1$ and $t = 0$ (and hence $d = 1$). Specifically, there exists an $\ell \in [s]$ such that $w(a_1, c_\ell) \ge \frac{1}{2}OPT$.*

Proof. Suppose by way of contradiction that no such ℓ exists. Then by the triangle inequality, $w(c_\ell, c_m) < OPT$ for all $\{\ell, m\} \in \binom{[s]}{2}$. But this contradicts the optimality of S, since it implies that every edge in S has weight strictly less than OPT. Thus we conclude that the set $S - T$ does indeed contain a vertex c_ℓ satisfying the statement of the lemma. □

Lemmas 6, 7, 8, 9, and 10 together imply Lemma 4, which in turn implies Theorem 5, stating that d-Greedy Augment has approximation ratio $(2k-2)/(k+d-2)$.

3. EXPECTED CASE PERFORMANCE

In this section, we make some observations about the expected characteristics of random instances of the remote-clique problem and the expected performance of d-Greedy Augment on these instances. In order to make the random graph models easier to analyze, we consider unweighted graphs (or equivalently weighted graphs in which the edge weight is always 0 or 1). Although this changes the problem because the triangle inequality is no longer satisfied, analyzing the expected behavior of this problem is a first step in understanding the expected behavior of the remote-clique problem. As we will see, it also provides some inuition for the design of our experiments in Section 4.

Let $G_{n,p}$ be a random graph on n vertices in which each edge is chosen independently with probability p. It has been proved that for any constant $\epsilon > 0$, if $k = (2+\epsilon)\log_{1/p} n$ then the probability of $G_{n,p}$ having a clique of size k approaches zero as n approaches infinity. (Thus we say that *almost every* graph does not have a clique of size k.) Conversely, it has been shown that if $k = (2-\epsilon)\log_{1/p} n$, then almost every graph $G_{n,p}$ has a clique of size k [1]. There are many similar results in the theory of random graphs, and the function $k = 2\log_{1/p} n$ is called a *threshold function* for the property of having a clique of size k.

In order to better understand the expected behavior of random instances of the remote-clique problem, we seek a threshold function for property $P_{k,r}$, defined as follows.

Definition 1. *We say that random graph $G_{n,p}$ has property $P_{k,r}$ if there exists a set V' of k vertices such that at least r edges in $G_{n,p}$ have both endpoints in V'.*

Note that $P_{k,r}$ is a generalization of the property of having a clique of size k since the two properties are equivalent if $r = \binom{k}{2}$. We have a partial result regarding this property, although we do not have a complete proof of a threshold function. To prove our result, we will need the following technical lemma, which is an example of one of the well-known *Chernoff bounds*. For a proof, the reader is referred to [10].

Lemma 11. *Let $B_{n,p}$ be a binomial random variable with parameters n and p. Then for any δ such that $0 < \delta \leq 1$,*

$$\Pr\left(B_{n,p} \geq (1+\delta)np\right) \leq e^{-np\delta^2/3}$$

We now proceed with the statement and proof of our result regarding property $P_{k,r}$.

Theorem 12. *Let $r = (1+\delta)\binom{k}{2}p$ for some $0 < \delta \leq 1$. Then for any integer $k \geq 3$, if $(k-1)p\delta^2 = 6(1+\epsilon)\ln n$ for some constant $\epsilon > 0$, then*

$$\Pr\left(G_{n,p} \text{ has property } P_{k,r}\right) \leq n^{-\epsilon}$$

Proof. Let random graph $G_{n,p}$ have vertex set V and edge set E. For each $V' \in \binom{V}{k}$, let $X_{V'}$ be a random variable defined as follows

$$X_{V'} = \begin{cases} 1 & \text{if there are at least } r \text{ edges in } E \text{ having both endpoints in } V' \\ 0 & \text{otherwise} \end{cases}$$

By a union bound, we have that

$$\begin{aligned} \Pr\left(G_{n,p} \text{ has property } P_{k,r}\right) &\leq \sum_{V' \in \binom{V}{k}} \Pr\left(X_{V'} = 1\right) \\ &= \binom{n}{k} \Pr\left(B_{\binom{k}{2},p} \geq r\right) \end{aligned}$$

By Lemma 11, then,

$$\Pr\left(G_{n,p} \text{ has property } P_{k,r}\right) \leq \binom{n}{k} e^{-\binom{k}{2}p\delta^2/3} \leq \left(\frac{ne}{k}\right)^k e^{-\binom{k}{2}p\delta^2/3} \tag{3.1}$$

where the second inequality comes from the easily proved estimate $\binom{n}{k} \leq (ne/k)^k$ (see [8]). Inequality (3.1) simplifies to

$$\Pr\left(G_{n,p} \text{ has property } P_{k,r}\right) \leq \left(n^{1+\frac{1-\ln k}{\ln n} - \frac{(k-1)p\delta^2}{6\ln n}}\right)^k$$

By substituting $6(1+\epsilon)\ln n$ for $(k-1)p\delta^2$, we get

$$\begin{aligned} 1 + \frac{1-\ln k}{\ln n} - \frac{(k-1)p\delta^2}{6\ln n} &= 1 + \frac{1-\ln k}{\ln n} - \frac{6(1+\epsilon)\ln n}{6\ln n} \\ &= -\epsilon + \frac{1}{\ln n}(1-\ln k) \\ &\leq -\epsilon \end{aligned}$$

where the last inequality follows from the fact that $k \geq 3$ and hence $(1-\ln k)$ is negative. We conclude that

$$\Pr\left(G_{n,p} \text{ has property } P_{k,r}\right) \leq n^{-\epsilon k} \leq n^{-\epsilon}$$

□

Note that the expected number of edges in any set of k vertices is $\binom{k}{2}p$. Therefore, if δ is a constant close to 0, then Theorem 12 states that if $(k-1)p\delta^2 = 6(1+\epsilon)\ln n$, then the

probability that there exists a set of k vertices that is significantly heavier than the average weight is small. In this case, then, randomly choosing any group of k vertices will give nearly an optimal solution on average. Thus, assuming that p is a constant, this problem is not very interesting when k is significantly greater than $\log n$.

Now that we have a result describing what happens when k is large, we turn towards examining what happens when k is small. Specifically, we ask how small k has to be to guarantee that Greedy Augment (d-Greedy Augment with $d = 1$) always finds a complete subgraph of k vertices on $G_{n,p}$. Towards this goal, we prove the following result.

Theorem 13. *Let E be the event that* Greedy Augment *returns a set of vertices with less than $\binom{k}{2}$ edges on $G_{n,p}$. Then for any constant $\epsilon > 0$*

$$p^{k'}(n - k') - \ln k' \geq \epsilon \ln n \Rightarrow \Pr(E) \leq n^{-\epsilon}$$

where $k' = k - 1$.

Proof. Let E_i be the event that on the i^{th} augmenting step, Greedy Augment chooses a vertex v' in which there is at least one vertex $v \in T$ such that the edge $\{v', v\}$ does not exist. Thus $E = \bigcup_{i=1}^{k} E_i$, and we have

$$\Pr(E) \leq \sum_{i=1}^{k} \Pr(E_i) = \sum_{i=2}^{k} \Pr(E_i) \leq (k-1)\Pr(E_k)$$

where the equality follows because $\Pr(E_1)$ is clearly 0, and the last inequality follows because $\Pr(E_1) \leq \ldots \leq \Pr(E_k)$. Event E_k happens exactly when for all vertices $v' \in V - T$, there exists a vertex $v \in T$ such that the edge $\{v, v'\}$ is not in the graph. The probability of this occuring for a single vertex v' is $1 - p^{k-1}$. Since there are $n - k + 1$ vertices in $V - T$, we therefore have that $\Pr(E_k) = (1 - p^{k-1})^{n-k+1}$. Substituting k' for $k - 1$, we have

$$\begin{aligned} \Pr(E) \leq k'(1 - p^{k'})^{n-k'} &\leq k' e^{-p^{k'}(n-k')} \\ &= n^{\frac{\ln k' - p^{k'}(n-k')}{\ln n}} \end{aligned}$$

But by assumption, we have

$$\frac{\ln k' - p^{k'}(n - k')}{\ln n} \leq \frac{\ln k' - (\ln k' + \epsilon \ln n)}{\ln n} = -\epsilon$$

which proves the claim. □

Note that the smaller k is, the larger the value of $p^{k'}(n-k') - \ln k$ is. Therefore, this theorem shows that if k is small enough relative to n, then the probability of Greedy Augment *not* finding a full clique on k vertices approaches 0 as n gets large.

To conclude this section, we describe one more result of a slightly different nature that is motivated by the following line of reasoning. Suppose we have a random graph model in which we know that there is at least one set of k "seed" vertices that is completely connected. Then a reasonable strategy for attempting to find this clique would be to choose a large number of sets of r vertices (where $r \leq k$) and proceed as Greedy Augment would from these sets, returning the best solution found. The hope with this strategy is that at least one of these sets of vertices would be completely contained in the seed clique and that with high probability Greedy Augment would perform optimally for this set of vertices by returning the seed clique itself.

A first step in asking whether such a strategy is feasible is to ask how big r must be to guarantee that with high probability Greedy Augment, when started with a set R of r vertices inside of a completely connected set K of k vertices, returns the set K itself. Along these lines, we show in the next theorem that if r is large enough relative to n and k then the probability that Greedy Augment will return a set of k vertices other than K approaches 0 as n approaches infinity. To do this, we first formally define the random graph model we use.

Definition 2. *Let $H_{n,p,K}$ be a random graph on n vertices where K is a k-element subset of the vertices and each edge e is chosen independently with probability p_e, where*

$$p_e = \begin{cases} 1 & \text{if both endpoints of } e \text{ are in } K \\ p & \text{otherwise} \end{cases}$$

We now show the following.

Theorem 14. *On random graph $H_{n,p,K}$, suppose that* Greedy Augment *is started with the set of r vertices R chosen randomly from K. Let E be the event that* Greedy Augment *returns a set of vertices different than K. Then for any constant $\epsilon > 0$,*

$$r = \log_{\frac{1}{p}}(n^{\epsilon}(n-k)(k-r)) \Rightarrow \Pr(E) \leq n^{-\epsilon}$$

Proof. For $r+1 \leq i \leq k$, let E_i be the event that Greedy Augment chooses a vertex $v' \notin K$ to be the next vertex to add when there are currently $i-1$ vertices in T. Then $E = \bigcup_{i=r+1}^{k} E_i$, and therefore

$$\Pr(E) \leq \sum_{i=r+1}^{k} \Pr(E_i) \leq (k-r)\Pr(E_{r+1})$$

where the last inequality follows because $\Pr(E_{r+1}) \geq \ldots \geq \Pr(E_k)$. For E_{r+1} to occur, there must be at least one vertex $v' \in V - K$ such that v' has an edge to every vertex in R. By a union bound, the probability of this is no greater than $(n-k)p^r$. Therefore, we have that $\Pr(E) \leq (n-k)(k-r)p^r$. But by assumption, $(n-k)(k-r) = n^{-\epsilon}/p^r$. Thus, we conclude that $\Pr(E) \leq n^{-\epsilon}$. □

4. EXPERIMENTAL RESULTS

In this section, we experimentally test the performance of d-Greedy Augment and related heuristics. At first we focus on the following four algorithms, although we will eventually introduce a new algorithm called Smart Augment.

- Random. This algorithm chooses a random subset of k vertices and returns the result. We include this in our experiments to make sure that our less trivial heuristics perform better than this.
- Greedy Augment. This algorithm is d-Greedy Augment with $d = 1$. Therefore it has a running time of $O(kn)$ and it guarantees an approximation ratio 2.
- 2-Greedy Augment. This algorithm is d-Greedy Augment with $d = 2$. It has a running time of $O(kn^2)$ and it guarantees an approximation ratio of $2 - \frac{2}{k}$. Although this is asymptotically equivalent to the ratio provided by Greedy Augment, we test it experimentally because it is conceivable that it performs significantly better than Greedy Augment in practice.
- All-Greedy Augment. For each vertex in $v \in V$, this algorithm runs Greedy Augment with the set T initialized with v. (Recall that in the standard version of Greedy Augment, the set T is initialized with an arbitrary vertex.) All-Greedy Augment returns the best solution found this way. Since All-Greedy Augment runs the algorithm Greedy Augment n times, the running time of All-Greedy Augment is $O(kn^2)$.

As we will see, the performance of these algorithms in practice seems to be much better than the approximation ratio of 2 might suggest, even on graphs whose edge weights do not obey the triangle inequality. In the experiements we run, even Greedy Augment does not ever return a solution whose value is worse than $\frac{3}{4}$ times the optimal solution. Therefore, because of its speed, Greedy Augment provides an attractive option for the task of approximately solving remote-clique. Nevertheless, it is not clear from our analyses earlier in this thesis how the more complicated algorithms 2-Greedy Augment and All-Greedy Augment perform relative to Greedy Augment. It is one of the goals of this section to answer which of these algorithms provides a better option if a quadratic time algorithm is acceptable for a given application.

Finding a random graph model to test these algorithms experimentally is a challenge. The straightforward model to choose would be $G_{n,p}$, but there are several problems with this choice. First, the edge weights in $G_{n,p}$ do not obey the triangle inequality. Although this does not make the problem any easier, it would be nice if at least one model that we test the algorithms on is consistent with the model from Section 2. We could make the edges satisfy the triangle inequality if we added 1 to the weight of every edge in $G_{n,p}$ (that is, if we made it so that every edge has weight either 1 or 2 and the probability that an edge has weight 2 is p). However, even using this model has problems. As shown by Theorem 12, for values of k much larger than $\log n$, the weight of an optimal solution is not significantly greater than that of a random solution. Thus, this graph model does not seem to be a useful benchmark for comparing our algorithms. Even for small values of k, there is still no efficient way to determine the value of the optimal solution in $G_{n,p}$.

To rectify these problems, we introduce a heavy "seed clique" in the random graph, as in the model $H_{n,p,K}$ of Section 3. In other words, we select k vertices at random and set the weight of edges between these vertices to be 2. However, this introduces the new problem that the vertices in the seed clique have a much higher degree than vertices outside of the seed clique (where the degree of a vertex v in a weighted graph is defined to be the sum of the weights of the edges incident to v), and thus the problem becomes easy to solve by the trivial algorithm that chooses the k vertices with highest degree. We can fix this by reducing the expected weight of edges that have one endpoint in the seed clique and the other outside of the seed clique. That is, if we lower the probability that such an edge has weight 2 by the right amount, then we can force the expected degree of all the vertices to be the same. Along these lines, we define the graph model $A_{n,p,k}$ formally as follows.

Definition 3. *Random graph $A_{n,p,k}$ is the graph on n vertices created as follows. First, a set of k random vertices is chosen to be the seed clique. Then, the weights of the edges are chosen independently to be either 1 or 2 such that the probability p_e that edge e has a weight of 2 is*

$$p_e = \begin{cases} 1 & \text{if both endpoints of } e \text{ are in the seed clique} \\ p & \text{if both endpoints of } e \text{ are outside the seed clique} \\ \frac{(n-k-1)(1+p)-2(k-1)}{n-2k} - 1 & \text{otherwise} \end{cases}$$

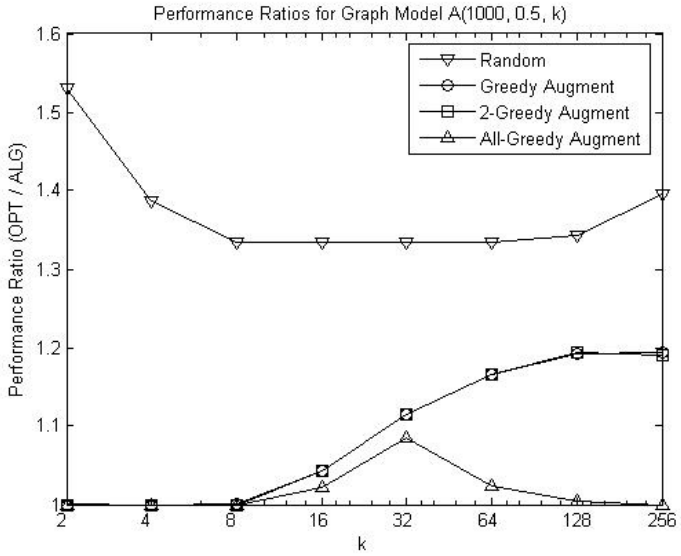
If $((n-k-1)(1+p)-2(k-1))/(n-2k)-1 < 0$ (which may happen for large k), then $A_{n,p,k}$ is not defined. Note that an optimal solution in $A_{n,p,k}$ has an average edge weight of 2. Also, since the expected weight of edge e is $1+p_e$, the expected degree of a vertex in the seed clique is

$$(k-1)\cdot 2 + (n-k)\cdot\frac{(n-k-1)(1+p)-2(k-1)}{n-2k} = \frac{(n-k-1)(n-k)(1+p)-2k(k-1)}{n-2k}$$

whereas the expected degree of a vertex outside of the seed clique is

$$(n-k-1)(1+p)+k\cdot\frac{(n-k-1)(1+p)-2(k-1)}{n-2k} = \frac{(n-k-1)(n-k)(1+p)-2k(k-1)}{n-2k}$$

Fig. 4.1: Performance of the algorithms on $A_{1000,0.5,k}$.



Thus we see that, as desired, the expected degree of every vertex is the same.

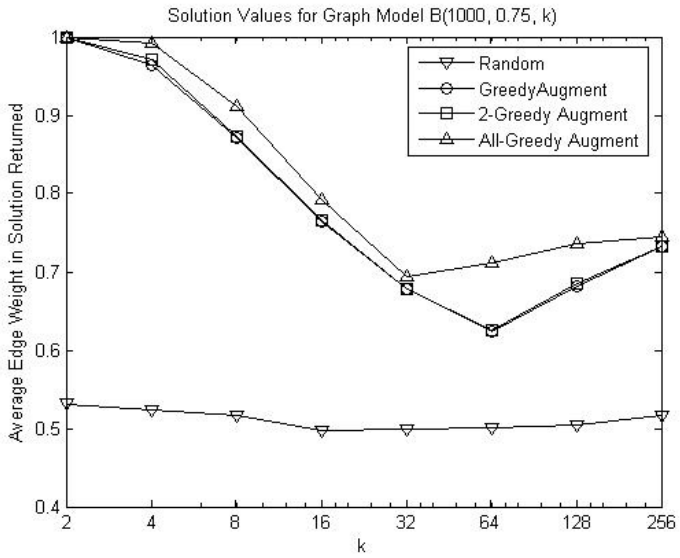
We tested the performance of the four algorithms listed above on instances of $A_{n,p,k}$. We set n to be 1000 and p to be $\frac{1}{2}$. We varied k from 2 to 256. For each value of k, we created 100 random instances of $A_{n,p,k}$ and ran all four algorithms on these instances. In Figure 4.1, we plot the average performance ratio from these experiments (the value of the optimal solution, $k(k-1)$, divided by the value of the solution returned by the algorithm) as a function of k.

All three of our heuristics (Greedy Augment, 2-Greedy Augment, and All-Greedy Augment) not only have performance ratios significantly better than 2, but, more importantly, they also perform significantly better than Random. Perhaps surprisingly, Greedy Augment and 2-Greedy Augment perform nearly identically on this graph model. All-Greedy Augment, on the other hand, performs much better than both Greedy Augment and 2-Greedy Augment, particularly as k gets large.

Although from this experiment it appears that All-Greedy Augment is the best algorithm, it is possible that its good performance is an artifact of the graph model $A_{n,p,k}$ because as k gets larger, it becomes increasingly unlikely that an edge with one endpoint in the seed clique and one outside has weight 2. For example, in $A_{1000,0.5,256}$, the probability of such an edge having weight 2 is approximately 0.24. Therefore, as k gets larger, it becomes less likely that if Greedy Augment begins with T completely contained in the heavy seed clique (which All-Greedy Augment is guaranteed to do in at least k iterations) it will ever choose a vertex outside of the seed clique.

To help ensure that the results from this experiment were based on more than the particular characteristics of the graph model $A_{n,p,k}$, we tested the performance of the algorithms on

Fig. 4.2: Performance of the algorithms on $B_{1000,0.75,k}$.



another graph model, called $B_{n,q,k}$. In this graph model, the weights of the edges are chosen independently and uniformly to be between 0 and 1. Again, we provide a heavy seed clique to make sure that the weight of an optimal solution is significantly greater than that of a random solution. A natural choice for the weight of the edges in the optimal solution would be 1, but this has the potential to make the problem too easy, since the weight of every edge in the optimal solution would be much higher than the average edge weight. Instead, we set the weight of the edges in the heavy clique to be q, for some $0.5 < q < 1$. Doing this means that we no longer know that the seed clique is in fact an optimal solution (and chances are that it will not be for small values of k). Since there is no good way to determine what the exact value of the optimal solution is, it is not possible to measure the exact performance ratio for this model. However, if we test the algorithms on the same instances of $B_{n,q,k}$, then we can still measure their relative performance by the value of the solution that they return. We define graph model $B_{n,q,k}$ more formally as follows.

Definition 4. *Random graph $B_{n,q,k}$ is the graph on n vertices created as follows. First, k vertices are chosen randomly to be in the seed clique. The weight of each edge with both vertices in the seed clique is set to q, and the weight of the remaining edges is chosen independently and uniformly to be between* 0 *and* 1.

Note that the edge weights in $B_{n,q,k}$ do not satisfy the triangle inequality, but this should not make the model any easier for an algorithm.

The results of a test of our algorithms on graph model $B_{n,q,k}$ is shown in Figure 4.2. Again, we set n to be 1000 and varied k from 2 to 256. We set q to be 0.75. For each value of k, we created 100 instances of $B_{n,q,k}$ and tested each of the four algorithms on these

instances. In Figure 4.2, which shows the average value of the solution returned by each of the algorithms, we see results that are mostly similar to those of our previous experiment. The major difference between these results and the results for model $A_{n,p,k}$ is that the gain in performance of All-Greedy Augment is not as pronounced for high values of k, although this is not unexpected given our previous discussion. However, just as in our previous experiment, all three of our heuristics perform significantly better than Random. Greedy Augment and 2-Greedy Augment perform nearly identically, whereas All-Greedy Augment consistently performs better than the other two algorithms.

Although All-Greedy Augment outperforms both Greedy Augment and 2-Greedy Augment, it is still a fairly simple algorithm, and it is natural to wonder if there is a "smarter" algorithm that performs better in practice than All-Greedy Augment. For the remainder of this section, we describe an algorithm called Smart Augment that attempts to be more careful than All-Greedy Augment about which vertices to choose.

One of the advantages of All-Greedy Augment seems to be that once Greedy Augment starts with a set of vertices inside an optimal solution, it is unlikely to add vertices outside of the optimal solution since the edge weights in the optimal solution will be much larger than the edge weights to vertices outside of the optimal solution. However, when T is still a small set, this algorithm can be "fooled" into choosing vertices outside of the optimal solution if they have a heavier weight to T than the vertices in the optimal solution. Smart Augment attempts to rectify this problem by first choosing a set of vertices that are likely to be optimal, namely, the union of T and the set of $k - |T|$ vertices in $V - T$ whose weight to T is highest (call this set T'). Then Smart Augment chooses a vertex that maximizes the average edge weight to $T \cup T'$ (instead of just the edge weight to T, as All-Greedy

Fig. 4.3: Performance of Smart Augment on $A_{500,0.5,k}$.

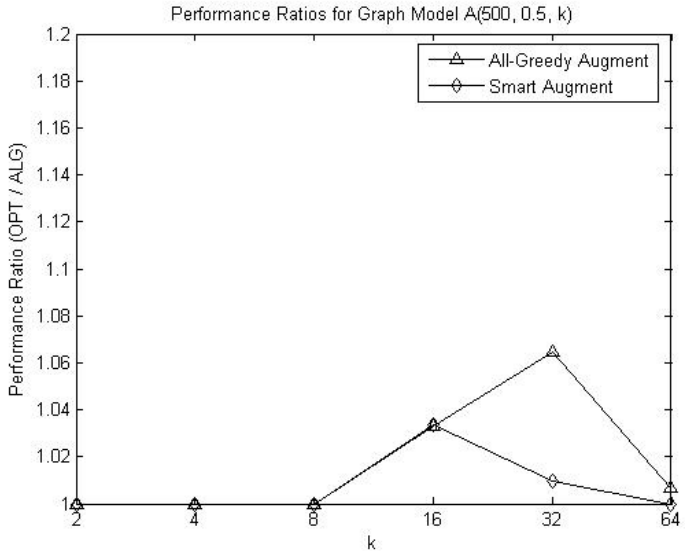
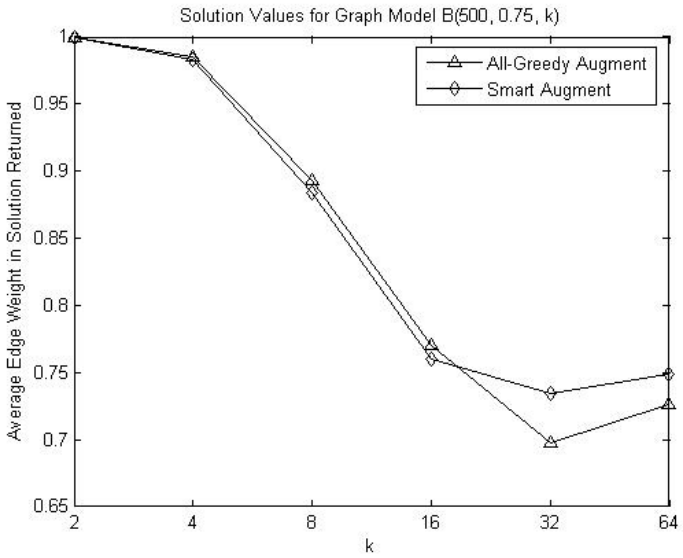


Fig. 4.4: Performance of SMART AUGMENT on $B_{500,0.75,k}$.



AUGMENT does). The idea behind this is that any random fluctuations that may make a vertex outside of the optimal solution look locally attractive will be "smoothed out" by choosing vertices based on their weight to a large set of vertices that is likely to overlap significantly with the optimal solution. As in ALL-GREEDY AUGMENT, SMART AUGMENT iterates through each vertex $v \in V$ and initializes the set T with v. It then returns the best solution obtained. Thus we are guaranteed that in at least k iterations, the set T is initially contained within an optimal solution.

An implementation of SMART AUGMENT is listed as Algorithm 2. Each iteration of the while loop takes $O(kn)$ time to compute the set T' and to determine the weight of each vertex to the set $T \cup T'$. Since there are $O(k)$ iterations of the while loop for each starting vertex v_0, the time for each iteration of the outer for loop is $O(k^2 n)$. Thus the overall running time of SMART AUGMENT is $O(k^2 n^2)$.

We tested the performance of SMART AUGMENT on graph models $A_{n,p,k}$ and $B_{n,q,k}$ and compared it to the performance of ALL-GREEDY AUGMENT on the same graph instances. Because of the slower running time of SMART AUGMENT (even without the extra factor of k the constants hidden by the asymptotic notation seem to be much higher), we set n to be 500 and only tested values of k up to 64. We also used only 20 problem instances for each value of k. The results of the performance of SMART AUGMENT on graph model $A_{n,p,k}$ are plotted in Figure 4.3, and the results of the performance of SMART AUGMENT on graph model $B_{n,q,k}$ are plotted in Figure 4.4. For graph model $A_{n,p,k}$, the two algorithms performed nearly identically, except for intermediate values of k, when SMART AUGMENT performed better. For graph model $B_{n,q,k}$, the results were similar. SMART AUGMENT performed only slightly worse than ALL-GREEDY AUGMENT for small values of k, and for larger values of

Algorithm 2 SMART AUGMENT

BestSolutionValue $\leftarrow -\infty$
for all $v_0 \in V$ **do**
  SolutionValue $\leftarrow 0$
  $T \leftarrow \{v_0\}$
  LastAdded $\leftarrow v_0$
  while $|T| < k$ **do**
    for all $v \in V - T$ **do**
      $\text{aug}(v) \leftarrow \text{aug}(v) + w(\textit{LastAdded}, v)$
    end for
    Find T', the set of the $k - |T|$ vertices v' in $V - T$ with the highest values of $\text{aug}(v')$
    BestVertexValue $\leftarrow -\infty$
    for all $v \in V - T$ **do**
      VertexValue $\leftarrow 0$
      for all $v' \in T \cup T'$ **do**
        VertexValue $\leftarrow$ *VertexValue* $+ w(v, v')$
      end for
      if $v \in T'$ **then**
        VertexValue $\leftarrow$ *VertexValue*$/(k-1)$
      else
        VertexValue $\leftarrow$ *VertexValue*$/k$
      end if
      if *VertexValue* $>$ *BestVertexValue* **then**
        BestVertexValue $\leftarrow$ *VertexValue*
        BestVertex $\leftarrow v$
      end if
    end for
    $T \leftarrow T \cup \{\textit{BestVertex}\}$
    SolutionValue $\leftarrow$ *SolutionValue* $+ \text{aug}(\textit{BestVertex})$
    LastAdded $\leftarrow$ *BestVertex*
  end while
  if *SolutionValue* $>$ *BestSolutionValue* **then**
    BestSolutionValue $\leftarrow$ *SolutionValue*
    BestSolution $\leftarrow T$
  end if
end for
return *BestSolution*

k, it performed a little bit better.

To summarize the results of this section, our experiments show that all of the heuristics based on d-Greedy Augment performed very well for the two random graph models we examined. Greedy Augment has the natural advantage of being a very fast algorithm that returns a solution that is guaranteed to be no worse than $\frac{1}{2}$ optimal and which in practice seems to do much better. However, if it is possible to spend more time, then the algorithm All-Greedy Augment is the natural choice because it seems to perform significantly better than Greedy Augment in practice with a slowdown of only a factor of n in running time. The algorithm 2-Greedy Augment should be avoided since it has the same running time as All-Greedy Augment, but performs the same as Greedy Augment. Finally, the algorithm Smart Augment performs slightly better than All-Greedy Augment, although it is much slower. Therefore it should only be used if efficiency is of little concern.

5. CONCLUSION

In this thesis, we have used the technique of factor-revealing linear programs to prove that d-Greedy Augment achieves an approximation ratio of $(2k-2)/(k+d-2)$. This improves the best-known approximation ratio of Greedy Augment from 4 to 2. Since we have shown that there are an infinite number of problem instances in which d-Greedy Augment returns a solution no better than $(k+d-2)/(2k-2) \cdot OPT$, we have completely characterized the worst-case performance of d-Greedy Augment.

We also examined how d-Greedy Augment performs in practice, both by providing some theoretical results regarding the expected performance of Greedy Augment on random graphs and by experiementally analyzing the behavior of d-Greedy Augment and related heuristics. Our experiments show that All-Greedy Augment, a simple modification of Greedy Augment, seems to perform better in practice than Greedy Augment. Therefore, if $O(kn^2)$ time complexity is acceptable for a given application, All-Greedy Augment provides an attractive choice for the algorithm designer.

There are several remaining open questions. First, there are no hardness of approximation results for the remote-clique problem. An interesting direction for future research would be to either find an algorithm that achieves an approximation ratio asymptotically better than 2 or to show that this is imposssible if $P \neq NP$. There are also unanswered questions from our analysis of the expected performance of Greedy Augment on random graphs. Each of the three theorems in Section 3 establishes a condition that is sufficient to guarantee with high probability that a property of Greedy Augment on a random graph holds. However, the theorems do not show that these conditions are *necessary* to guarantee these properties. Establishing such necessary conditions, and thereby determining whether our functions are true threshhold functions, is another interesting direction for future research.

BIBLIOGRAPHY

[1] Béla Bollobás. *Random Graphs*. Academic Press, London, 1985.

[2] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI '99: Proceedings of the third symposium on Operating systems design and implementation*, pages 173–186, Berkeley, CA, USA, 1999. USENIX Association.

[3] Barun Chandra and Magnus M. Halldorsson. Approximation algorithms for dispersion problems. *J. Algorithms*, 38(2):438–465, 2001.

[4] U. Feige, D. Peleg, and G. Kortsarz. The dense k-subgraph problem. *Algorithmica*, 29(3):410–421, 2001.

[5] R. Hassin, S. Rubinstein, and A. Tamir. An approximation algorithm for maximum dispersion. *Operations Research Letters*, 21:133–137, 1997.

[6] Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *J. ACM*, 50(6):795–824, 2003.

[7] M.J. Kuby. Programming models for facility dispersion: the p-dispersion and maxisum dispersion problems. *Geographical Analysis*, 19(4):315–329, 1987.

[8] Jiri Matousek and Jaroslav Nesetril. *Invitation to Discrete Mathematics*. Oxford University Press, Oxford, 1998.

[9] Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. Adwords and generalized on-line matching. In *FOCS '05: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 264–273, Washington, DC, USA, 2005. IEEE Computer Society.

[10] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, 2005.

[11] S. S. Ravi, D. J. Rosencrantz, and G. K. Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42:299–310, 1994.