**University of Alberta**

# An Empirical Examination of the $k$-Center Problem In Graphs

by

**Robert David Beck** ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 1998

0-612-34337-5

University of Alberta

Library Release Form

**Name of Author:** Robert David Beck

**Title of Thesis:** An Empirical Examination of the $k$-Center Problem In Graphs

**Degree:** Master of Science

**Year this Degree Granted:** 1998

[illegible]

Robert David Beck
8101 160 Street,
Edmonton, Alberta,
Canada, T5R 2G9

**Date:** Oct 7 98

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **An Empirical Examination of the $k$-Center Problem In Graphs** submitted by Robert David Beck in partial fulfillment of the requirements for the degree of **Master of Science.**

[illegible]

Joe Culberson

[illegible]

Lorna Stewart

[illegible]

Maziar Shirvani

**Date:** Oct 5 1998

# Abstract

Informally, the k-center problem is that of choosing k locations on a weighted graph so that the maximum cost to any vertex from the closest of the k locations is minimized. We review existing complexity results, as well as exact and approximate algorithms for the k-center problem in graphs. This includes the fact that the problem is NP-hard. We examine the work done by exact algorithms when applied to randomly generated problem instances to gain an understanding of where the problem may be difficult to solve, and how good we may expect approximate solutions to be. Finally, we present new heuristics for approximating k-center which show significant improvements over previously published solutions on a test bed of randomly generated instances.

For Cathryn and Calvin
and for Chenelle's remaining sanity

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Overview and Motivation

A problem arising in many practical applications is that of facility location; we wish to determine where to locate facilities that provide services to other locations. We may wish to locate these facilities in a manner that a maximum property (such as distance) with respect to the other locations is minimized. This type of problem is an instance of the *k-center* problem in graphs. In informal terms, the *k-center* problem is that of fixing *k* locations on a graph so that the maximum cost to get to anywhere in the graph from the closest of the *k* locations is minimized. The cost may be biased to account for the "importance" of the place being served. The usual example used to describe the *k-center* problem is that of locating emergency response facilities on a network of roads so that the the maximum response time, biased by the population served, is minimized. The *k* locations chosen are referred to as a *k-center*. The maximum cost we would incur to get from a location in the *k-center* to anyplace else in the graph is referred to as the *radius* of the *k-center*.

Our focus on this problem was initiated by a problem from the forest industry locating equipment bases for logging. This involved locating sizable centers on large graphs. In this problem, the equipment bases could be located only at particular points, which in turn correspond to vertices of a graph in the *k-center* problem.

In the general case, the *k-center* problem can be difficult to solve exactly, and as we will see, will have an exponential worst case bound on the running time of an exact algorithm. A worst case bound however, does not guarantee that all instances of the problem take exponential time, nor does it tell us how well we may do with an attempt to obtain an approximate solution to the problem in less time. Several techniques have been published on obtaining approximate solutions to the problem.

## 1.2 Background

The *k-center* problem itself is very closely related to the *r-dominating set* problem. In the *r-dominating set* problem we fix a maximum cost allowed, and allow the number of source points to vary. We then wish to find the smallest number of source points so that the maximum cost of getting to any point on the network from a source does not exceed our fixed maximum cost.

The *k-center* and *r-dominating set* problems both involve *minimizing* a *maximum* property in the solution, involving the distance from one point to another. Garey and Johnson [4] refer to the *k-center* problem as the *Min Max Multicenter* problem. These problems, and others that involve minimizing a maximum property are often referred to as *bottleneck* problems [8] [6] [3]. The *k-center* and *r-dominating set* problems are closely related to the *dominating set* problem [4] [6] [13] [5].

In the literature on the *k-center* and related problems, many authors use subtly, and sometimes significantly different definitions of the problem. The *k-center* problem is frequently referred to as the *p-center* problem [10] [11] [16]. We choose to use $k$ in this work to avoid confusion with $p$ which we use to represent a probability.

## 1.2.1 Complexity Classes P and NP

We refer frequently to work from complexity theory, and in particular to the theories of NP-completeness. Garey and Johnson [4] provide an in depth treatment of the subject, and a large reference of problems and their classifications. We provide the basic background here:

**Definition 1**

*A decision problem is any question that can be answered as "yes" or "no".*

Given an optimization problem (such as *k-center*) we can express it as a related decision problem. Informally, The *k-center* decision problem is that of deciding if a *k-center* of radius $r$ exists in a graph. See Definition 12 in section 2.1 for a formal definition of the decision problem.

Decision problems are the type of problems traditionally studied by the theory of computation. When examining optimization problems it is typical to study the decision version of the problem [15] [4]. This is because it is often possible to use the decision version of the problem to solve the optimization version of the problem, and the decision versions of the problem are typically easier to work with, as most existing work on complexity classes deals with decision problems.

We use $P$ to refer to the class of all decision problems which can be solved by a polynomial time algorithm. We use $NP$ to refer to the class of all decision problems for which if $x$ is a positive (yes) instance of the problem, there exists a $certificate$ of length bounded by a polynomial in the size of $x$ which can be used to check that $x$ is a positive instance of the problem in polynomial time.

If a problem is in $P$, then it is also in $NP$. The $certificate$ of a positive instance of a problem in $P$, is simply the instance itself. By virtue of its membership in $P$, we know we can verify a positive instance of the problem by applying the polynomial time algorithm to solve the problem to the instance, and this will verify it.

Consider the *k-center* decision problem. A *certificate* for a positive instance of the problem is a *k-center* of radius $\leq M$. To verify a certificate $X$, we verify that $|X| = k$ and that the radius is less than $M$. This can be easily done in polynomial time. Thus we see that *k-center* is a member of class $NP$. It is not clear if there is a polynomial time algorithm to solve instances of the *k-center* decision problem, as there may be $\binom{n}{k}$ [1] certificates that we would have to test in the above manner if we did not have the certificate given to us.

Another important concept is that of a polynomial transformation. Given two decision problems, $Q_1$ and $Q_2$, $Q_1$ is said to be polynomially transformable to $Q_2$ if given $x$ an instance of $Q_1$ we can construct in polynomial time an instance $y$ of $Q_2$ such that $x$ is a positive instance of $Q_1$ if and only if $y$ is a positive instance of $Q_2$. The important implication is that given an algorithm for $Q_2$ we could use it to solve $Q_1$ by applying our algorithm for $Q_2$ to the polynomial transformation $y$. So, if there is a polynomial time algorithm for $Q_2$, there is a polynomial time algorithm for $Q_1$.

This leads to the concept of $NP$-completeness. A decision problem $Q$ is said to be $NP$-complete if $Q$ is a member of $NP$, and all other problems in $NP$ polynomially transform to $Q$. An NP-complete problem has one very important property, in that if a polynomial time algorithm exists for it, then we know that a polynomial time algorithm exists for all members of the class $NP$. The usual way of showing a problem to be $NP$-complete is by polynomially transforming a known $NP$-complete problem to it, which shows that the problem is $NP$-hard, and proving the problem to be a member of $NP$. Many problems have been shown $NP$-complete. The reader is referred to [4] for initial explicit proofs of $NP$-completeness, and for large lists of known $NP$-complete problems. A problem $H$ is said to be $NP$-hard if all other problems in $NP$ polynomially transform to $H$. An $NP$-hard problem is not $NP$-complete unless the problem itself is a member of class $NP$.

If a polynomial time algorithm exists for an $NP$-complete problem, we then every member of $NP$ is also a member of $P$, and we would then have $P = NP$. It is widely believed (although not proven) that $P \neq NP$, and that no polynomial time algorithms exist for $NP$-complete problems [4] [15] [5].

$NP$-completeness results tell us when problems could be hard to solve exactly. They tell us where a general case of a problem may be difficult to solve. They do not tell us that all instances of a problem are difficult to solve. An NP-completeness result does not preclude finding solutions easily given restricted or specific instances of the problem.

## 1.3 Synopsis

In this thesis, we perform a review of the existing literature on the *k-center* problem and some related problems. From this we see that the *k-center* problem will be NP-hard to solve, or to approximate with a *k-radius* guaranteed to be better than twice that of an optimal solution.

We then perform some initial experimentation to familiarize ourselves with the problem and how

[1] which is exponential if $k$ is not fixed

exact algorithms can perform on randomly generated instances with varying properties, We verify that we can generate random problem instances that won't be trivially easy to solve.

We present new algorithms for approximating *k-center*. Using our initial observations and review of the problem, we explore the idea of using new heuristics to choose vertices in a greedy algorithm based on properties of vertices that are nearby. We call these heuristics *Ball Heuristics* and we demonstrate that they can be used to obtain better approximate results in many cases than previously published algorithms for approximating *k-center*.

Chapter 2 contains background and review of the existing literature on algorithms for *k-center*. In this chapter we define the *k-center* problem formally, as well as as a number of similar and related problems showing the relationship between them. We present references and summaries of previous work done on the problem.

Chapter 3 contains the description and experimental results from initial experiments used to familiarize ourselves with where the problem may be difficult to solve exactly and where it may be easy, as well as some more in depth examination of what sort of distribution of work we can expect from an exact algorithm on randomly generated problem instances.

Chapter 4 contains our development of new variations of existing approximation algorithms for the *k-center* problem. We present algorithms that show improvement over existing algorithms in the literature for the problem on many classes of random problem instances.

Chapter 5 contains a summary of our conclusions and suggestions for further work on the *k-center* problem.

# Chapter 2

# Background and Review

We formally define the *k-center* problem and related problems that we will be using.

## 2.1 Definition of the *k-center* problem

**Definition 2** (*k-set*)

*A k-set is a set of cardinality k.*

**Definition 3** (*undirected weighted graph*)

*In an undirected weighted graph $G = (V, E, w, l)$, every vertex $v$ is assigned a non-negative real number $w(v)$ called the weight of $v$, and every edge $uv$ is assigned a positive real number $l_G(uv)$, called the length of $uv$. We use $n = |V|, m = |E|$ throughout. All graphs are considered undirected unless otherwise specified throughout.*

**Definition 4** (*path*)

*A path $p(u, v)$ from $u$ to $v$ in a graph is a sequence of vertices $v_1 \cdots v_j$ where $v_1 = u$, $v_j = v$, and $v_i v_{i+1} \in E | 1 \leq i \leq j$.*

*The length $l(p)$ of a path $p$ is $l(p) = \sum_{i=1}^{j-1} l(v_i v_{i+1})$.*

**Definition 5** (*distance*)

*The distance between two vertices in a graph $G$, $d_G(u, v)$ is the length of the shortest path from $u$ to $v$ in $G$, i.e. $\min(l(p)) \forall p(u, v)$.*

*The distance between a vertex $v$ and a set $X \subset V$ is the length of a minimal length path from $v$ to any vertex in $X$ i.e. $d_G(v, X) = \min\{d(v, x) | x \in X\}$*

The *k-center* and *r-dominating set* problems are as follows:

**Definition 6** (*radius*)

*Given a weighted graph $G$, and a k-set$X \subset V$ the radius $\eta$ induced by k-set $X$ is:*

$$\eta_G(X) = \max_{v \in V}\{w(v)d_G(v, X)\}$$

**Definition 7 (optimal *k-center* and *k-radius*)**

*an optimal k-center for a weighted graph $G$ is a k-set$X \subset V$ where:*

$$\eta^* G(X) = \min_{(Y \subset V||Y|=k)} \eta(Y)$$

*such an $\eta^*$ is called the k-radius of $G$.*

**Definition 8 (*k-center* optimization problem)**

*The k-center optimization problem is given a weighted graph $G$, and an integer $k$ find the k-radius of $G$ and an optimal k-set.*

**Definition 9 (*absolute k-center*)**

*A point on $G$ can be either a vertex $v \in V$, or a point on an edge $e \in E$ with e regarded as a line segment of length $l_G(uv)$.*

*Given $G$ and $k$, the absolute k-center problem is to find a k-set $Y$ of points on $G$ such that $\eta(Y)$ is minimized.*

**Definition 10 (*dominating set*)**

*A dominating set in a graph $G = (V, E)$ is a set of vertices $X \subseteq V$ such that for all vertices $v \in V - X$, there is an $x \in X$ for which $xv \in E$.*

*Given a graph $G = (V, E)$, the dominating set problem is to find a minimum cardinality set $X \subseteq V$ that is a dominating set in $G$.*

**Definition 11 (*r-dominating set*)**

*Given $G$ and $r$, the r-dominating set problem is to find the smallest cardinality set $Z \subset V$ such that that $\eta(Z)$ does not exceed $r$.*

**Definition 12 (*k-center* and *r-dominating set* decision problems)** *The k-center decision problem is Given weighted graph $G$ and integer $M$, is there is a subset of vertices $X \subset V$ of size $k$, such that $\eta(X)$ is less than or equal to $M$. This is exactly equivalent to the r-dominating set decision problem of whether there exists a r-dominating set of radius $M$ and cardinality $k$.*

## 2.2 Complexity of Exact Solutions

Kariv and Hakimi [10] show the NP-completeness of the *k-center* problem by reducing from the *dominating set* problem, shown to be NP-complete by Garey and Johnson [4]:

**Theorem 1** *Given graph $G = (V, E)$, and given integer $k$, $1 < k < n$ as input, the problem of determining the existence in $G$ of a dominating set of cardinality $\leq k$ is NP-complete even when $G$ is planar and of maximum vertex degree 3. [4]*

Given the above, we may prove the following [10]:

**Theorem 2** *The problem of finding a k-center or absolute k-center is NP-hard even when the graph is a vertex unweighted planar graph of maximum vertex degree 3, where all edges are of length 1.*

**Proof:**

Suppose we can find a *k-center* $X$ with *k-radius* $r$ in $G$. Then there exists a dominating set of cardinality $\leq k$ if and only if $r \leq 1$. Thus *dominating set* is polynomially reducible to *k-center* and finding a *k-center* is NP-hard. Similarly, let $X$ be an *absolute k-center*, then if $r > 1$ no dominating set of cardinality $\leq k$ can exist. For $r \leq 1$, we replace every non-vertex point in $X$ by the closest vertex to it. Since the length of every edge is 1, we now have a *k-center* of $r \leq 1$, and a dominating set of cardinality $\leq k$ exists, therefore *dominating set* is also polynomially reducible to finding an *absolute k-center*, and finding an *absolute k-center* is NP-hard.[10]

## 2.3 Exhaustive Algorithms

In [10], Kariv and Hakimi present an algorithm to solve the *absolute k-center problem*, as well as an exhaustive algorithm for a *k-center*.

**Exhaustive Algorithm**

**Step 0** To find a *k-center*, we check every *k-set* $X \subseteq V$, and choose that which minimizes $\eta(X)$.

Since $\eta(X)$ can be computed in $O(nk)$, the complexity of the exhaustive algorithm is $\binom{n}{k}O(nk) = O(\frac{n^{k+1}}{(k-1)!})$. [10]

For an *absolute k-center*, we must consider not only vertex points, but points on edges. The algorithm of Kariv and Hakimi [10] for an *absolute k-center* limits the number of points we must consider to a finite number by finding only *suspected points* on edges which may be candidates for a *k-center*. Kariv and Hakimi show that there are $O(mn^2)$ of these points[10], and show the complexity of their algorithm to be $O(\frac{m^k n^{2k-1}}{(k-1)!} \lg n)$ when exhaustively searching these points.

These algorithms are not exponential in $n$ or $m$, but rather are exponential in $k$. For fixed $k$ these algorithms are polynomial. In practice however, these algorithms will be of little use for $k$ and $n$ of any significant size.

## 2.4 Complexity of Approximate Solutions

We define the *$\alpha$-approximate k-center* problem as the problem of finding a center of size $k$ in $G$, whose radius, $r$, satisfies $r \leq \alpha r^*$, where $r^*$ is the *k-radius* $G$.

When examining methods for obtaining approximate solutions to a problem it is helpful to define an approximation problem in terms of the worst case deviation of the approximate solution from the optimal solution. In such a manner we can define the approximation problem and determine how the complexity of the approximation problem will differ from that of the original. The reader is referred to [15] and [5] for many examples of this.

Hsu and Nemhauser [8] prove the following two theorems (in the context of Theorem 2):

**Theorem 3** *The $\alpha$-approximate k-center problem is NP-hard for $\alpha < 2$.*

**Theorem 4** *The $\alpha$-approximate absolute k-center problem is NP-hard for $\alpha < \frac{3}{2}$.*

The first is proven by noticing that since $r^*$ is an integer and thus any value of $\alpha$ less than 2 will solve the *dominating set* problem. The second is true since for any $\alpha < \frac{3}{2}$, we can move the approximate solution points to the closest vertex, and again, we solve *dominating set.*

## 2.5 Approximation Algorithms

### 2.5.1 Plesnik

Hochbaum and Schmoys [7] produced the first $\alpha = 2$ 2-approximation for the *k-center* problem for complete graphs. This was later generalized to general connected graphs and to a 2-approximation of the *absolute k-center* problem by Plesník [16].

When considering the *k-center* problem on graphs, algorithms in the literature frequently require that the weights on the edges of a graph satisfy the *triangle inequality*, which is as follows:

**Definition 13** (*triangle inequality*) *The edges of a weighted graph $G = (V, E, w, l)$ are said to satisfy the triangle inequality if for every edge $uv \in E$, $d_G(u,v) = l_G(uv)$; that is, no path between two vertices may be shorter than the length of the edge between them.*

**Definition 14** (*distance graph*)

*The distance graph of a weighted graph $G = (V, E, w, l)$ is a complete weighted graph $D_G = (V', E', w', l')$ where $V' = V$, $w'(v) = w(v)$ for all $v \in V'$, and $l_{D_G}(uv) = d_G(u,v)$ (or infinite if $u$ and $v$ are not connected in $G$).*

Plesník [16] notes that many authors define the *k-center* problem only on a complete graph whose edges satisfy the triangle inequality [7] [9], but in fact these algorithms are applicable to the general case, where our initial graph may not have these properties. This is based on the observation that if we compute a distance graph from the original graph, this graph will have the triangle inequality, and finding a *k-center* in the distance graph is equivalent to finding one in the original graph.

We can compute the distance graph $D_G$ from $G$ in $O(n^3)$) time. Since for any *k-set* $X$, $\eta_{D_G}(X) = \eta_G(X)$, we may apply any *k-center* algorithm designed for complete graphs whose edges satisfy the triangle inequality to any weighted graph by computing the distance graph $D_G$ from $G$ and applying the algorithm to $D_G$.

Plesník's algorithm for the *k-center* problem is as follows:

**Plesnik Heuristic Center**

Input: Weighted graph $G$, integer $k$.

Output: *k-set* $S'$, lower bound $r'$.

**Step 1** Construct $D_G$ if not given, and sort the $n(n-1)$ multi-set of weighted distances $d_G(u,v)(w(v)$ for $u, v \in V$ into a non-decreasing sequence. Delete duplicates to reduce this sequence to an increasing sequence $\{f_1 < f_2 < \cdots < f_q\}$ .

**Step 2** Find $r'$, the least value of $r \in \{f_1 < f_2 < \cdots < f_q\}$ for which procedure RANGE yields an output $S$ with $|S| \leq k$.

**Step 3** Augment $S$ arbitrarily to a set $S'$ of $k$ vertices. Output $S'$ and $r'$, stop.

**Procedure RANGE**

: Input: Weighted graph $G$, integer $k$, weighted distance $r$.

: Output: Set $Y$, a center of radius $\leq 2r$ in $G$.

**Step 0** At first, all vertices of $G$ are unlabeled: $Y \leftarrow \phi$.

**Step 1** While there are unlabeled vertices, choose an unlabeled vertex $u$ of maximum weight, let $Y \leftarrow Y \cup \{u\}$. Label the vertex $u$ and every unlabeled vertex $v$ such that $w(v)d(u,v) \leq 2r$.

**Step 2** Output $Y$.

By using a binary search in Step 2, it is sufficient to apply RANGE only $O(\log(n))$ times. RANGE itself is of complexity $O(n^2)$. If $D_G$ is given to us, Step 1 in Heuristic CENTER can be performed by sorting in $O(n^2 \log(n))$ making the complexity of the algorithm $O(n^2 \log(n))$ [16].

Plesník's RANGE procedure picks from the vertices in the graph for the *k-center* problem. It is changed for the ***absolute k-center problem*** by picking instead from the $O(mn^2)$ points which can be candidates for solution points in the ***absolute k-center*** problem as described in Kariv and Hakimi

[10]. This then increases the complexity of the algorithm for the *absolute k-center* problem to $O(mn^2 \log(n))$

Each algorithm is shown to be a 2-approximation for the problem, which in the case of the *k-center* problem is the best possible assuming $P \neq NP$ [8] [16].

**Theorem 5**

*For any $r > 0$, if there exists a k-set $X \subseteq V(G)$, with $\eta(X) \leq r$, then procedure RANGE from Plesnik Heuristic Center finds a set $Y \subseteq V(G)$ with $|Y| \leq k$ and $\eta(Y) \leq 2r$.*

Proof:

Let $X$, where $\eta(X) \leq r$, be a set consisting of vertices $\{x_1 \cdots x_k\} \subset V(G)$. Let $S_1 \cdots S_k$ be any partition of $V$ satisfying $x_i \in X_i$ and $\forall v \in S_i, w(v)d(x_i, v) \leq r$.

Now consider $Y$, the set returned by procedure RANGE. We know by construction from the algorithm that for $v \in V(G)$, $w(v)d(Y, v) \leq 2r$, so $\eta(Y) \leq 2r$. Consider any $u$, chosen at Step 1 of RANGE. Let $S_i$ be the set containing $u$. For every other *unlabelled* vertex $v \in S_i$ we have $w(v) \leq w(u)$. By the triangle inequality we then have $w(v)d(u,v) \leq w(v)[d(u,x_i) + d(x_i,v)] \leq w(u)d(x_i,u) + w(u)d(x_i,v) \leq 2r$. Once $u$ is added to set $Y$, no other vertex from $S_i$ may be added, as all other vertices in $S_i$ will be labelled. This proves that RANGE will add at most 1 vertex from each $S_i$ to the set $Y$, therefore guaranteeing $|Y| \leq k$.

**Theorem 6**

*Plesnik Heuristic Center is a 2-approximate solution for the k-center problem.*

Proof:

The *Plesnik Heuristic Center* algorithm finds the smallest weighted distance $r$ from the graph for which RANGE returns a set of size $\leq k$. If we assume that *Plesnik Heuristic Center* is not 2-approximate, then there must be a distance $r^*$, the radius of an optimal center, which is less than the $r$ found. Obviously, $r^*$ must be a weighted distance from the distance graph, meaning that when tried with radius $r^*$ RANGE did not return a set $S$ with $|S| \leq k$, or $r*$ would have been chosen instead of $r$. This contradicts Theorem 5, proving that *Plesnik Heuristic Center* is 2-approximate.

Theorem 5 will not hold if RANGE does not choose $u$ of maximum weight at step 1. As an example, consider a 4-clique where all edges are of length 1, two of the vertices are of weight 1, and the other two vertices are of weight $Z$, where $Z > 2$. This graph has two distinct weighted distances in it (1 and $Z$), and a 2-center of radius 1 (The two vertices of weight $Z$). Any other 2-center will have a radius of $Z$. If RANGE ever chooses a non maximum weight vertex at step 1, it will not be able to find a size 2 set for any radius below $Z$.

### 2.5.2 Martinich

Martinich [9] takes a slightly different approach to the *k-center* problem (he does not consider the *absolute k-center* problem). Martinich's Algorithms *SEQ* and *BIN* have a worse worst-case bound than *Heuristic Center*, but a better average case error performance over the test cases examined in [9].

Martinich's algorithms take a "vertex closing" approach to finding a *k-center*. The algorithm initially takes all the vertices as the potential center, and then attempts to remove vertices from the set making up the center while keeping $\eta$ small. It works by iteratively constructing subgraphs $G_1, G_2 \cdots$ and at each step finding as small a dominating set as possible, building up larger sets each time. We note that finding minimum dominating sets is also NP-hard[4]. Martinich's algorithms greedily construct "small" but not optimal dominating sets.

Martinich ran and compared his algorithm (*SEQ*) to Hochbaum and Schmoys heuristic center[7]. The results indicate that *SEQ* performed well in the average case for graphs with a ratio of $\frac{k}{n} > 0.3$. *SEQ* performed well on the test cases in [9], with average error (from the optimal *k-radius*) in the range of several percent, as opposed to the average error of 30-50% for Hochbaum and Schmoys (*Plesnik Heuristic Center*) on the same problems.

The drawbacks of Martinich's algorithm are that it is quite slow ($O(n^4)$), and did not perform well on $\frac{k}{n} < 0.3$, or on large graphs ($n > 100$). The Hochbaum and Schmoys algorithm was much faster on large graphs [9]. This leads us to believe that while Martinich's algorithm may be useful for locating centers that are large relative to the size of the graph, it may not be very useful for smaller centers.

### 2.5.3 Dyer and Freize

Dyer and Freize present a different heuristic algorithm in [2]. The algorithm of Dyer and Freize is as follows:

**Dyer and Freize Heuristic Center**

Input: Weighted graph $G$, integer $k$.

Output: *k-set* $S$

**Step 0** Construct $D_G$ if not given; $S \leftarrow \theta$

**Step 1** Choose $v_1 \in V$ of maximum weight. Add $v_1$ to $S$. Set $D(v) = w(v)d(v_1, v)$ for each $v \in V$.

**Step 2** $i = 1$; While $i < k$

- Determine $v_{i+1}$ such that $D(v_{i+1}) = \max_{v \in V} D(v)$. Add $v_{i+1}$ to $S$.
- Set $D(v) = \min\{D(v), w(v)d(v_{i+1}, v)\}$ for all $v \in V$.

**Step 3** Output $S$.

Independent of Step 0, *Dyer and Freize Heuristic Center* takes $O(nk)$ operations to complete[2]. *Dyer and Freize Heuristic Center* is shown to give a $\min(3, 1+\alpha)$-approximate solution where $\alpha = \frac{\max_{v \in V}(w(v))}{\min_{v \in V} w(v)}$[2].

## 2.6 *k-center* in trees

The *k-center* problem is solvable in polynomial time on trees. Kariv and Hakimi [10] present several algorithms for solving the problem on trees. The general algorithm presented for trees is based on the same observation as made in [16], that the optimal *k-radius* must be one of the weighted distances from the distance graph, and therefore there are at most $n(n-1)$ possible values for the optimal *k-radius*.

Kariv and Hakimi show that an *r-dominating set* can be found in a tree in $O(n)$ time [10]. It is then shown that a *k-center* can be found in a tree in polynomial time by the following algorithm:

***k-center* of vertex weighted tree**

Input: Weighted tree $T$, integer $k$.

Output: *k-set* $S'$.

**Step 1** Calculate the $O(n^2)$ possible values for $\eta(X)$, the radius of an optimal *k-set* in $T$. Sort and delete duplicates to arrange in a nondecreasing sequence $f_1 < f_2 < \cdots < f_q$.

**Step 2** Find $r'$, the least value of $r \in \{f_1 < f_2 < \cdots < f_q\}$ for which we can find a *r-dominating set* $S$ in $T$ with $|S| \leq k$.

**Step 3** Augment $S$ arbitrarily to a set $S'$ of $k$ vertices. Output $S'$.

The complexity of the algorithm is shown to be determined by Step 1, which will require $O(n^2 . \lg n)$ operations [10].

## 2.7 The *k-center* problem in random graphs

Lacking data from a specific application of the k-center problem to focus on, to examine the behaviour of algorithms for *k-center* we resort to randomly generated graphs.

In [6], Hochbaum examines the *dominating set* problem applied to random graphs, and uses the following definition of the "K-center" problem: [1]

[1] We capitalize K here to indicate that the problem as defined in Definition 15 is different from the *k-center* problem as we define it in Definition 8

**Definition 15** *("K-center" problem) As a decision problem, the "K-center" problem is: Given $G$, is there a subset of vertices of size $K$, such that the maximum weight* **on the edge** *between each vertex and the nearest vertex in the subset is less than or equal to $r$.*

Hochbaum examines the ease of finding optimal and approximate solutions to the *dominating set* problem and the "K-center" problem in random graphs. The results are based on the analysis of *dominating set*, and concludes:

> In certain random environments these problems become "ridiculously easy" to solve asymptotically. In fact, the solution amounts to picking indiscriminately a feasible solution. [6].

Unfortunately, the difference between the definition of the "K-center" problem used in [6] and the *k-center* problem will make the results from [6] inapplicable to *k-center*.

The *dominating set* problem is analyzed for a family of random graphs defined in [6] as: $\mathcal{G}_{n,p} = (V_n, E_{n,p})$. A graph in class $\mathcal{G}_{n,p}$ is a graph on $n$ vertices where each edge belongs to $E_{n,p}$ with probability $p$. The "K-center" problem is then defined on a complete graph with the edge weights associated with each pair of locations assigned as independent and identically distributed random variables.

In this context, Hochbaum proves the following results in [6].

**Theorem 7** *[6]*

*There is, almost surely, a dominating set of size $K in \mathcal{G}_(n,p)$ if $p \geq p_u(n,K)$, where $p_u(n,K) = O(1 - \frac{1}{(n-K)^{\frac{1}{K}}})$*

*There is, almost surely, no dominating set of size $K$ in $\mathcal{G}_(n,p)$ if $p \leq p_l(n,K)$, where*

$p_l(n,K) = O(1 - \frac{1}{(n-K)^{\frac{1}{K-1}}})$

From Theorem 7 it is shown:

**Corollary 1** *[6]*

*For any fixed value of $0 < p < 1$, for a graph from $\mathcal{G}_(n,p)$:*

*There is no dominating set of size $K$ for $K = o(\ln n)$ almost surely.*

*There is a dominating set of size $K$ for $K = \Omega(\ln n)$ almost surely.*

*Whenever $K = c\ln(n)$ there is a dominating set of size $K$ almost surely whenever $p \geq p_u(n, c\ln n)$, and furthermore, $\lim_{n\to\infty} p_u(n, c\ln n) = \lim_{n\to\infty} p_l(n, c\ln n) = 1 - e^{\frac{-1}{c}}$*

Using definition 15 of the "K-center" problem from [6], answering the question of whether there is a solution for the "K-center" decision problem for radius $r$ amounts to finding a dominating set of size $K$ or less in a random graph derived from a complete graph with randomly assigned edge weights, where we remove all edges for which the weight exceeds $r$. Such a random graph is a random graph from class $\mathcal{G}_{n,p}$ where $p$ is the probability of a vertex having weight assigned (in the original complete graph) less than $r$[6]. Therfore, the analytical bounds established for *dominating set* in [6] hold for "K-center".

These results from [6] are not applicable to the *k-center* decision problem in a useful way. To see why, we first recall the *k-center* decision problem (Definition 12), and we define a Weighted Distance Graph:

**Definition 16** (*weighted distance graph)*

*The weighted distance graph of a weighted graph $G = (V, E, w, l)$ is a complete directed weighted graph $W_G = (V', E', w', l')$ where $V' = V$, $w'(v) = 1$ for all $v \in V'$, and the length of every edge $uv$ in $W_G$ is the weighted distance $w(v)d(u, v)$ from $G$, or an infinite length if $u$ and $v$ are not connected in $G$.*

**Theorem 8**

*Every instance of the k-center decision problem is equivalent to an instance of the dominating set problem on the corresponding Weighted Distance Graph.*

Proof:

We construct the weighted distance graph $W_G = (V', E', w', l')$ from $G = (V, E, w, l)$, where $V' = V$, For every $(u, v) \in E'$ $l'(u, v) = w(v) * d(u, v)$, and for all $v' \in V'$, $w'(v') = 1$.

Eliminating all edges the weights of which exceed $r$ from $W_G$ creates a graph that has a dominating set of size $k$ if and only if the answer to the *k-center* decision problem in $G$ is affirmative.

Unfortunately, we can not extend the *dominating set* results of [6] to the *k-center* problem. It is required for the analysis of *dominating set* in [6] that edge weights of the graph in $\mathcal{G}_{n,p}$ are assigned as independent and identically distributed random variables[6]. For the "K-center" problem, where we examine only the *edge length* between two vertices, this will be true. However, for the *k-center* problem, in which the *shortest path length* is used rather than the *edge length*, this will not be the case. The probability of the shortest path between two vertices having a particular length is not dependent solely on the probability of an edge being assigned a particular weight, but potentially also on the weights assigned to all other edges in the graph. Our weighted distance graph $W_G$ will not have the edges assigned independently. This means that $W_G$ will not be a random graph from class $\mathcal{G}_{n,p}$, so we may not directly use the results from [6] on the more general case of the *k-center* problem.

The "K-center" problem does establish an upper bound on a *k-center* in the sense that a solution for the "K-center" problem of radius $r$ means that we will certainly have a *k-center* solution of radius less than or equal to $r$. The reverse is not true. The lack of a solution for "K-center" of radius $r$ does not mean that a solution for *k-center* of radius $r$ does not exist. To see this we consider a class of graphs where we assign random edge lengths to a complete graph as length 3 with probability 0.5, and length 1 with probability 0.5. For a graph of significant size, with high probablilty a graph from this class will have a diameter of 2[14]. In this class of graphs a *k-center* and a "K-center" of radius 1 are equivalent, since there are no paths of length less than 1, and all results of [6] apply to both problems. If we consider a radius 2, the bounds for where a "K-center" may exist are unchanged (since "K-center" solutions may only have radius 3 or 1 in this class of graphs), yet in a graph of diameter 2, every vertex will be a *k-center* of size 1, radius 2. Thus we see there may be significant differences in where solutions can exist for the two problems.

Nikoletseas and Spirakis also give analysis for *dominating set* in random graphs in [13]. They focus their analysis on vertex-unweighted random graphs where an edge is assigned between two vertices with the probablility $p = \frac{1}{2}$. It is shown for this class of graphs ($\mathcal{G}_{(n,p)}$, with $p = \frac{1}{2}$) that:

- The probability of existence of dominating sets of size less than $\log n$ tends to zero as $n$ tends to infinity
- Dominating sets of size $\lceil \log n \rceil$ exist almost surely.
-

The paper also presents several simple algorithms which will approximate *dominating set* based on a greedy or repeated trials approach.

Central to the analysis of *dominating set* in [13], as in [6] is the ablility to express the probablity of two vertices being connected independently, i.e. $\forall v_1, v_2, v_3 \in V, P\{v_1v_2 \in E\} = P\{v_1v_2 \in E | v_2v_3 \in E\}$ We know that any set of size $k$ in the graph will be a dominating set if and only if all other $n-k$ vertices of the graph are connected to one of the $k$ vertices. If connectivity is independent as we have described above, we know that each vertex not in our *k-set* will not be dominated by the set with probability $p^k$, so the probability that a given *k-set* will be a dominating set will be $(1-p^k)^{n-k}$.

The above analysis doesn't work well when we consider *r-dominating set* or *k-center*. On such a random graph, a path of a particular length from one vertex to another will be dependent on how the vertices are connected, that is, $P\{d(u,v) \leq r\} = P\{(l(u,v) \leq r \vee \exists w, d(u,w) + l(w,u) \leq r\}$ The probability of a vertex not in a *k-set* being greater than distance $r$ from any vertices in the *k-set* will be conditional on the existence of all edges in the graph. We do not have a simple independent probability that we can express easily in such an analysis.

## 2.8 Related Problems

## 2.9 k-median

One closely related centering problem is the k-medians problem. The k-medians problem differs from the *k-center* problem in the goal when selecting the *k-set*. Rather than minimizing the maximum weighted distance of any vertex in the graph from the chosen set, the goal is to minimize the sum of all weighted distances in the graph from the set.

**Definition 17 (k-median)**

*Given a weighted graph $G$ and integer $k$, the k-median problem is to find a k-set $X \subset V$, such that the function:*

$$\kappa(X) = \sum_{v \in V} \{w(v)d(v, X)\}$$

*is minimized.*

*A k-median of $G$ is any optimal k-set $X$.*

The k-median problem is examined by Kariv and Hakimi in [11]. The problem is proved NP-complete in [11] by reduction to ***dominating set*** in the same manner as in [10]. An $O(n^2k^2)$ algorithm for finding a k-median of a tree is given in [11].

# Chapter 3

# Performance of standard algorithms

## 3.1 Existing Solutions on Random Graphs

### 3.1.1 Motivation

We are interested in several things:

1. We would like to know if we can generate random problem instances that appear appropriate to test against, and are not trivially easy for an approximation algorithm to solve exactly.
2. We would like understanding of where it is difficult to find an exact solution and where it is not. That is, we would like a first cut at identifying where the difficult regions of the problem are.
3. We would like an understanding of where the approximate solutions from the *Plesnik Heuristic Center* algorithm are good, and where they are not.

As an exploratory step we will examine the performance of an algorithm for an exact solution and an algorithm for an approximate solution to the *k-center* problem over a class of random problem instances. We will vary certain parameters of the random problem instances to see how these parameters cause the solutions and performance of these algorithms to vary.

### 3.1.2 Algorithms

An exact algorithm was needed both as a reference point and to see how effective one can be on our randomly generated problem instances. With *k-center* being NP-hard we are reasonably confident that such an algorithm will have an exponential worst case running time, but we do not yet know how well such an algorithm could perform when not in the worst case. Solutions from an exact algorithm will serve as a basis to compare against the approximate solutions for our initial test cases. In this manner we hope to gain an empirical idea of how well the approximation algorithms perform in practice. For an exact algorithm we implemented *Depth First Center.*

*Depth First Center* is a depth first search for an exact solution to the *k-center* problem. The algorithm recursively visits vertices in an ordered sequence, and examines all solutions that arise both from including and excluding the chosen vertex in a center. The order in which the vertices are selected for visiting may be altered according to several rules. A branching rule that determines whether to examine solutions with the vertex in the center first, or out of the center first may be specified. The algorithm will prune its search based on an upper bound.

```
Function RecursiveCenter (G,n,k,In,Out,Best,BestSet,Vseq,LB,InFirst)

Input: Graph G of order n. Integer k, size of center desired. Sets In and Out of vertices in/out of
    the center so far. Radius Best, the best radius so far. Set BestSet, the best k-set found so
    far. Sequence of vertices Vseq. LB, a lower bound on radius based on vertices excluded so far
    in Out. Branching rule InFirst.

Output: Assigns Best and BestSet if a better set is found, returns best radius found using vertices
    chosen from Vseq

Step 1 if (LB > Best)
        prune (return infinite radius).
    endif

Step 2 if (In is of cardinality k, or Out of cardinality n-k)
        compute radius of In; update Best and BestSet if necessary: return computed radius.
    endif

Step 3 Pick the next vertex v from Vseq. Set NewOut = Out ∪ v, and set NewIn = In ∪ v.
    Compute NewLB as the greater of LB and the shorted weighted distance to v from any vertex
    not in NewOut.

Step 4 if (InFirst)
        InRad=RecursiveCenter(G,n,k,NewIn,Out,Best,BestSet,Vseq,LB,InFirst)
        OutRad=RecursiveCenter(G,n,k,In,NewOut,Best,BestSet,Vseq,NewLB,InFirst)
    else
        OutRad=RecursiveCenter(G,n,k,In,NewOut,Best,BestSet,Vseq,NewLB,InFirst)
        InRad=RecursiveCenter(G,n,k,NewIn,Out,Best,BestSet,Vseq,LB,InFirst)
    endif

Step 5 Return the smaller of InRad and OutRad
```

**Depth First Center**

*Input:* Graph G of order n, Integer k, A Vertex ordering strategy, and Infirst; a branching strategy.

*Output:* Optimal *k-radius* (Best) and *k-center* (BestSet).

**Step 0** In, Out, BestSet are empty, Best is Infinite.

**Step 1** Preorder vertices according to our selection strategy in Vseq,

**Step 2** Best=RecursiveCenter(G, n, k, In, Out, Best, BestSet, Vseq, 0, InFirst)

**Step 3** Output Best and BestSet.

We chose to run *Depth First Center* using three combinations of vertex selection strategy and branching strategy:

**Max first, In first:** Pick vertices by decreasing vertex weight, searching possibilities that include the chosen vertex before searching possibilities that exclude it.

**Min first, Out first:** Pick vertices by increasing vertex weight, searching possibilities that exclude the chosen vertex before searching possibilities that include it.

**Random Selection, In first:** Pick vertices in a random order, searching possibilities that include the chosen vertex before searching possibilities that exclude it.

The choice of strategies for *Depth First Center* was based on intuition and early observation. Intuitively, we hope that including vertices of max weight will guide the search to an optimal or near optimal value early on, thereby enabling the search to do more pruning. Similarly, we hope that excluding vertices of minimum weight will also induce early pruning. As a reference point, the third case tests the strategy of including a random vertex.

### 3.1.3 Problem Instances

We require a class of random graphs over which to test the performance of the algorithms. Both [6] and [13] use the uniform model of a random vertex-unweighted, edge-unweighted graph for their analysis; that is, the class of graphs $\mathcal{G}_{n,p} = \{V_n, E_{n,p}\}$, where a graph $G$ in $\mathcal{G}_{n,p}$ is a graph on $n$ vertices where each edge belongs to $E_{n,p}$ with probability $p$.

We will need to define a a different model in which we can produce graphs of varying edge weights, and varying vertex weights, as the *k-center* problem is defined on graphs with weights both on the vertices and the edges.

**Definition 18**

$\mathcal{G}_{n,VW,EW,PV,PE} = (V, E, w, l)$

$|V| = n, |E| = m$

*$VW$ is a sequence of vertex weights*

*$EW$ is a sequence of edge weights*

*$PV$, and $PE$ are sequences of probabilities, where $\sum_{p \in PV} = 1$, and $\sum p \in PE \leq 1$.*

$|VW| = |PV|, |EW| = |PE|$

*A Graph $G$ in $\mathcal{G}_{n,VW,EW,PE,PV}$ is a weighted graph on $n$ vertices. For all $v \in V$, $w(v) = VW_i$ with probability $PV_i$. For all $uv \in E$, $l(uv) = EW_i$ with probability $PE_i$, and is infinite (a non-edge) with probability $1 - \sum_{p \in PE}$.*

As a preliminary set of experiments, graphs were constructed as random graphs $\mathcal{G}_{n,VW,EW,PV,PE}$ (18) where the parameters were varied as in the following table. Initially five graphs were generated with each combination of parameters for a total of 30 graphs. This sample size was inappropriately small and was changed to 25 graphs with each combination of parameters for a total of 150 graphs. Edge weights used were $[1 \cdots 10]$ for all graphs.

| class | n | VW | PV | PE |
|---|---|---|---|---|
| 1 | 30 | $[1 \cdots 30]$ | uniform $p = \frac{1}{30}$ | uniform $p = 0.01$ |
| 2 | 30 | $[1 \cdots 30]$ | uniform $p = \frac{1}{30}$ | uniform $p = 0.05$ |
| 3 | 30 | $[1 \cdots 30]$ | uniform $p = \frac{1}{30}$ | uniform $p = 0.1$ |
| 4 | 30 | $[1]$ | 1 | uniform $p = 0.01$ |
| 5 | 30 | $[1]$ | 1 | uniform $p = 0.05$ |
| 6 | 30 | $[1]$ | 1 | uniform $p = 0.1$ |

The above data was chosen to generate graphs that varied over a wide range of path lengths between vertices, and a wide range of vertex weights. Since the length of paths in the graph could be varied by varying either the values used to weight the edges, or the probabilities for weighting the edges, we chose for this experiment to vary the edge probabilities for a small (1..10) sequence of edge weights uniformly. The edge probabilities were chosen deliberately so that the low end (0.01 for each of 10 weights) would give low connectivity in the graph sizes used, and at the high end (0.1 for each of 10 weights) the graph would be complete.

For each of the generated graphs, four algorithms were run:

1. *Plesnik Heuristic Center*
2. *Depth First Center* using *Max First, In First*
3. *Depth First Center* using *Min First, Out First*
4. *Depth First Center* using *Random Selection, In First*

On each of the 300 graphs, each algorithm was run for the following values of $k$, looking for approximate (in the case of *Plesnik Heuristic Center*) or optimal (for the *Depth First Center* variants) *k-center*.

**30 node graphs** : 2,3,4,5,6,7,8,8,9,10,15,20

Resource usage was recorded, both in CPU time used, and in the number of nodes searched by the *Depth First Center* variants. CPU usage was capped at 10 minutes per run in order to abandon long runs. This 10 minute cap was not encountered on 30 node graphs. Experiments were also run over similar classes of 50 node graphs to confirm the trends seen in the 30 node results. The 50 node results did indicate similar trends to the 30 node results, however, some runs timed out, making meaningful numerical comparison of results for the timed out values impossible.

### 3.1.4 Observations

*Plesnik Heuristic Center* **approximation performance**

Given that the sample size is only over 25 graphs, these results should be considered merely representative. While within a class of graphs the variance of the size of center found was not high, as we will see the variance in the amount of work done was very high.

The *Plesnik Heuristic Center* algorithm is relatively quick. This is to be expected as the cost of *Plesnik Heuristic Center* is bounded above by the time taken to compute the distance graph of the initial graph[16]. The *Depth First Center* algorithm also computes the distance graph ahead of time, and therefore incurs the same setup cost. This could be bypassed by precomputing and saving this for each graph, but was not done for this experiment, consequently, the *Plesnik Heuristic Center* always ran as fast as the quickest *Depth First Center* searches.

The *Plesnik Heuristic Center* algorithm gives a lower bound and an approximate *k-center*. The *k-radius* of the approximation ranged between near optimal to twice optimal depending on the graph class and the size of center. This is as expected, given that the algorithm is guaranteed to produce an approximation of no worse than twice the optimal *k-radius*[16]. Figure 3.1 and Figure 3.2 show the mean optimal vs. mean approximate *k-radius* values for 30 node vertex weighted graphs with vertex weights assigned uniformly from [1..30]. In Figure 3.1 we see that the approximate solution of *Plesnik Heuristic Center* remains significantly larger than the optimal until the value of $k$ approaches 15, half the size of the graph. At large values of $k$, the approximate *k-radius* very closely matches (or matches) the optimal *k-radius*. Figure 3.2, shows a very similar trend, but with a much smaller *k-radius*. The difference in the experiments is that the edge probabilities for each weight went from 0.01 in Figure 3.1, to 0.1 in Figure 3.2. The result of the increased edge probabilities will be a more connected graph having correspondingly shorter paths, and thus a smaller *k-radius* on average.

Contrasting to this is the behavior observed in Figure 3.3, and Figure 3.4, where the vertices are all of weight 1. In Figure 3.3, each edge weight is assigned with probability 0.01, and we see results that differ from the previous case, as the approximate solution does not approach the value of the optimal solution for large values of $k$. Figure 3.4 shows on this is still the case with a denser graph. This experiment also produced results that seemed to step from one value to the next. This is due to the graph being relatively dense (50% probability of an edge) with no vertex weights. As a result there are are only very short paths of integer weight in the graph, and the radius of any *k-center* can only be one of several small values.

We see a trend here that vertex weights will help the *Plesnik Heuristic Center* algorithm obtain good centers as $k$ becomes large relative to $n$, whereas in absence of vertex weights the approximate solution does not approach the value of the optimal as rapidly. We do know that at $n = k$ the approximate and optimal solutions will be the same, so in the vertex unweighted graphs there must be some higher values of $k$ at which the approximate solution will be optimal.

While *Plesnik Heuristic Center* is still guaranteed to be 2-approximate on vertex unweighted graphs[16], the heuristic of choosing a maximum weight vertex first will correspond to choosing a random vertex on a vertex unweighted graph, or a graph where all vertices are of the same weight.

### Work done to find an optimal *k-center* using *Depth First Center*

As we expect from the complexity of the exhaustive search algorithm (2.3), the work done by *Depth First Center* should rise quite rapidly with the value of $k$. In absence of any means to prune or restrict the search, an exhaustive search algorithm would search $\binom{n}{k}$ nodes to find the optimal. $\binom{n}{k}$ is maximal at $k = \frac{n}{2}$. The pruning done in *Depth First Center* seems to impact the search done substantially in this respect, with the largest amount of search being done at values of $k$ significantly less than $\frac{n}{2}$. Additionally, the work done is not symmetrical about $k = \frac{n}{2}$, indicating that *Depth First Center* is able to prune much better as $k$ gets larger. Figures 3.5 and 3.6 show the results using Max First/In first *Depth First Center* on 30 node graphs with vertex weight chosen uniformly from [1..30]. Figure 3.8 and Figure 3.7 show the results for Min First/Out first *Depth First Center* on vertex unweighted 30 node graphs.

The standard deviation of the nodes expanded and search times with *Depth First Center* was quite high, In the more difficult regions of the problem it was frequently larger than the mean, Thus the mean isn't very useful to us. Most often this was due to a large search time for some graphs and relatively short ones for most others - this is obviously the case at k=7 in Figure 3.6, with a max of nearly 600000,and a median value that shows very close to 0 when presented on a scale relative to this one outlying maximum value.

The relative performance of the *Depth First Center* heuristics of Max First/In First, as compared to Min First/Out First is interesting. For vertex weighted graphs, including maximum weight vertices seems to work much better judging from the max and median values of nodes expanded. This is seen in Figure 3.9, for a 30 node vertex weighted graph of low edge probabilities, and is more pronounced in Figure 3.10, where the edge probability is much higher, and the graph is correspondingly denser.

The superior performance of Max first/In first as a strategy for *Depth First Center* in these cases is likely due to the tendency for it to make *correct* decisions about vertices being in or out of the center early, thereby enabling it to search to an optimal value early on, and in many cases, prune off much of the search that would otherwise be done. With the vertices weighted from [1..30], it is very likely that the higher weight vertices are in fact in an optimal center.

## 3.2 A more in depth look

### 3.2.1 Description

Previously, we noted that the nodes expanded by *Depth First Center* had a large variance in the regions where the mean number of nodes expanded was high. While we could see a large variance in the data, the sample size of 5 was far too small to obtain a good idea of what the actual distribution of the work done is.

In general, if the work done by *Depth First Center* to solve the problem is of high variance, we may be able to obtain a solution faster on average by using a technique of randomly restarting, or multiple independent searches. On the other hand, if the variance is lower such a technique may not be as successful.

We are therefore interested in gaining an understanding of the general shape of the distribution we can expect on the number of nodes expanded by *Depth First Center* to solve the problem in the difficult region

We previously found the *Max First / In First* strategy to work best for our vertex weighted graphs, and for the vertex unweighted graphs, a random selection strategy worked as well as anything else. We would like to examine the distribution for sparse graphs, and for dense graphs, both vertex weighted, and vertex unweighted. We will use the *Max First / In First* strategy for vertex weighted graphs, and a random vertex selection strategy for the vertex unweighted graphs.

Therefore, we will test *Max First / In First* and *Random Choice / Out First* on a large number of graphs from the following classes:

1. $\mathcal{G}_{30,[1..30],[1..10],[\frac{1}{30}..\frac{1}{30}],[0.01..0.01]}$;
2. $\mathcal{G}_{30,[1..30],[1..10],[\frac{1}{30}..\frac{1}{30}],[0.1..0.1]}$;
3. $\mathcal{G}_{30,[1],[1..10],[1],[0.01..0.01]}$;
4. $\mathcal{G}_{30,[1],[1..10],[1],[0.1..0.1]}$;

From each run we will collect data on both the number of nodes expanded to completion, as well as the number of nodes expanded until the optimal was actually found.

The *Max First / In First* strategy is deterministic, and will always produce the same results on any one graph. However, the *Random Choice / Out First* strategy is not, so multiple runs on the same graph can produce different results. Since we are using randomly generated problem instances, we can obtain different results for a problem in the same class either due to two different randomly generated problem instances, or two runs of an algorithm that contains randomness. Consequently, We will take a subset of 1/10th of our graphs above at random, and run the *Random Choice / Out First* strategy 10 times on each of them, to compare results against running one run on each of the entire sample. In this manner we hope to see if there are significant differences in results due to differences in the randomly generated problem instances as compared to the randomness from the randomized algorithm.

Additionally, we know from previous experimentation that the work done by *Max First / In First* may be widely distributed. We will select some graphs that produce extreme results for *Max First / In First* and compare and contrast running multiple *Random Choice / Out First* runs over them, as compared to running *Random Choice / Out First* on the entire sample. In this manner we hope to see if there are any significant differences in results due to some (as yet unidentified) special structure(s) of these problem instances that proved difficult for *Max First / In First* to solve.

### 3.2.2 Observations

#### Distribution of Work Done

The general shape of the distribution of work done over multiple runs in all cases looked similar to "heavy tailed" profile as described by Gomes, Selman, McAloon, and Tretkoff in [1], that is, a large number of short run times, tapering off rapidly to a long trail of a few large run times. Figure 3.11 shows the frequency of nodes expanded for one run of *Random Choice / Out First* on 300 graphs from class #3. Figure 3.12 shows one run of *Random Choice / Out First* and *Max First / In First* on 300 graphs from class #1. *Max First / In First*on class #2 and *Random Choice / Out First* on class #4 were similar. Many runs finish relatively quickly, with a few extreme cases taking much more search to complete.

#### Multiple Runs vs. Multiple Graphs

The distribution of search was examined for multiple runs on a small number of graphs, as compared to one run on a larger number of graphs. Figure 3.13 and Figure 3.14 show the frequency of the search expanding a particular amount of nodes using *Random Choice / Out First* on vertex unweighted graphs. In both figures 300 runs were done, both as 1 run each on 300 graphs, and 30 runs on 10 graphs chosen at random from a the original 300. Figure 3.13 shows results from class #3, Figure 3.14 from class #4.

We can see that this distribution of nodes expanded is very similar for the case of one run on 300 graphs, and to 10 runs on 30 graphs. This seems to indicate that at least for *Random Choice / Out First* the randomized nature of the algorithm will give as much variation in search time as will the variations in the randomly generated graphs within this class.

### 3.2.3 Extreme Results from *Max First / In First*

To see what effect properties of the graph had on the search time distribution, 5 expensive (300,000 to 900,000 nodes expanded using *Max First / In First*) and 5 inexpensive (< 3000 nodes expanded using *Max First / In First*) graphs from the original sample of 300 graphs from class #1 were chosen, 60 runs of *Random Choice / Out First* were done on each of the graphs, and the results compared against each other and the results of using *Random Choice / Out First* on the entire sample.

In Figure 3.15 we see the results of *Random Choice / Out First* on the 5 expensive graphs, as compared to the entire sample. In Figure 3.16 we see the 5 inexpensive graphs as compared to the entire sample from class #1. Figure 3.17 shows both plotted together. From this it can be seen that the graphs that were difficult for *Max First / In First* seem to be more difficult (with more frequent longer run times) for *Random Choice / Out First*, and the graphs that were easy for *Max First / In First* seem to also be easier for *Random Choice / Out First*. There must be some as yet unidentified properties to these particular graphs making the problem easier or harder to search than the entire sample.

### 3.2.4 Finding Optimal vs. Proving Optimal

In all the samples, significantly more work was done by *Depth First Center* to prove optimality than was done to finding the optimal. For example, in Figure 3.18, we see the search used to find the optimal *k-radius* and the search used to prove optimality for all graphs in a 300 graph sample of vertex unweighted graphs using *Random Choice / Out First*. On over 2/3 of the sample *Random Choice / Out First* finds the optimal very quickly, when compared to the full search. Figure 3.19 shows the same comparison for vertex weighted graphs using *Max First / In First*. Here, the difference is not so great, with the optimal usually being found in about half the search taken to prove the optimal, whereas when using *Random Choice / Out First* on the same graphs, seen in Figure 3.20, the distribution looks more like the vertex unweighted *Random Choice / Out First* case.

### 3.2.5 Conclusions

The difficult region of the problem lies in the smaller center sizes, not at $\frac{n}{2}$. Once $k$ becomes large enough relative to $n$, the problem becomes much easier to solve exactly. The difficulty rises sharply with $k$, then falls gradually as we approach $\frac{n}{2}$.

The *Plesnik Heuristic Center* algorithm approximation result approached the optimal for large values of $k$ on the vertex weighted graphs studied. Those are also the values for which the problem becomes easier to solve exactly. For smaller values of $k$, the *Plesnik Heuristic Center* algorithm approximation usually produced an approximate *k-radius* of 1.5 times the optimal. The approximation from *Plesnik Heuristic Center* approached twice the optimal value for large $k$, as the vertex weight selection heuristic from *Plesnik Heuristic Center* was not able to distinguish vertices more likely to be in the center.

Including Max weight vertices first is the better *Depth First Center* strategy for vertex weighted graphs, but including vertices first seems to be riskier in the case where no weights are available.

The *Depth First Center* search will always find the optimal solution if run to completion. *Depth First Center* can frequently find a better solution than *Plesnik Heuristic Center* in a similar or slightly longer time on these small graphs, but then takes much more time to search to completion and terminate with an optimal value.

The distribution of work done searching for an exact solution shows us that frequently a solution will tend to be found relatively quickly compared to the average amount of work done over a large sample. On the vertex unweighted graphs this distribution is very similar for multiple runs on a few graphs or one run on many graphs.

On hard and easy graphs for *Max First / In First Depth First Center Random Choice / Out First* run on the same graph exhibited a corresponding shift in the distribution of the run times This indicates that some graphs are generally easier or harder for a variety of search algorithms.

This sort of distribution indicates that this problem may be attacked by a technique that would allow us to explore different ordering of the vertices simultaneously, or with some type of restarting mechanism. An algorithm of multiple independent trials of *Depth First Center* with different vertex orderings, or one which used some form of random restarting on a different vertex ordering should give us better average-case performance.

### 3.2.6 Further Experiments

This experiment has brought to light several further issues to examine:

How do other distributions of vertex weights aside from unweighted and uniformly distributed from $1 \cdots n$ affect the algorithms? All our samples used edge weights of [1..10] and a small fixed probability for each weight. How do varying vertex weights affect the algorithms?

*Depth First Center* frequently found optimal or near optimal solutions very early on in its search. Can we establish tighter bounds on the solution to increase our confidence in the optimality of a solution without having to perform a search to completion?

Can the *Plesnik Heuristic Center* algorithm lower bound be used to our advantage in a search for an optimal solution?

Can we improve on the average case performance of the *Plesnik Heuristic Center* 2-approximate algorithm in practice?

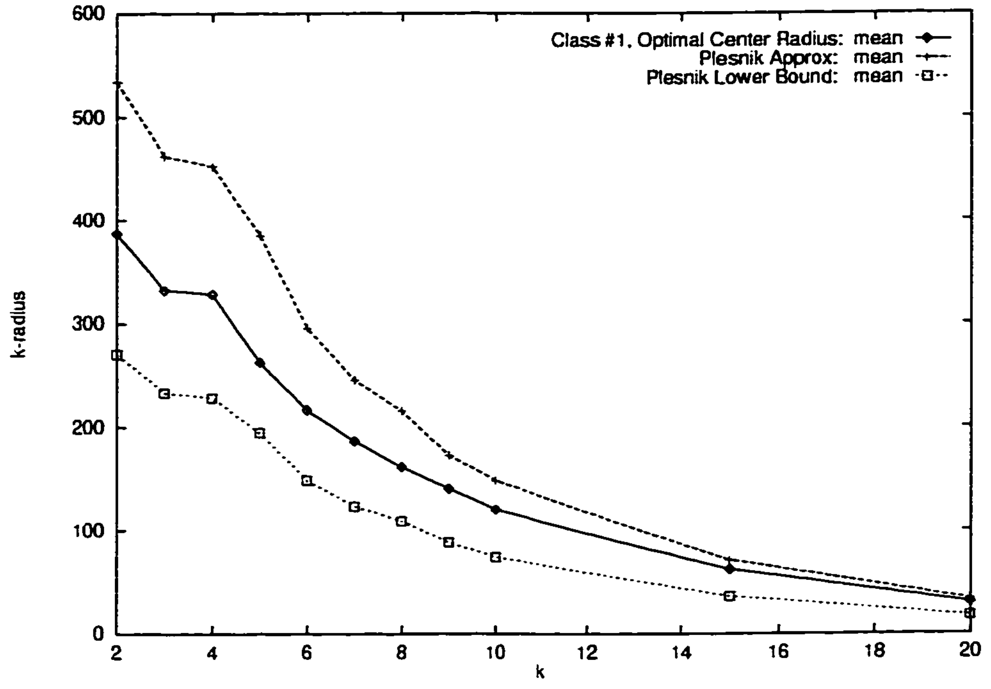

Figure 3.1: Optimal Center Radius vs Approximate Bounds, class #1

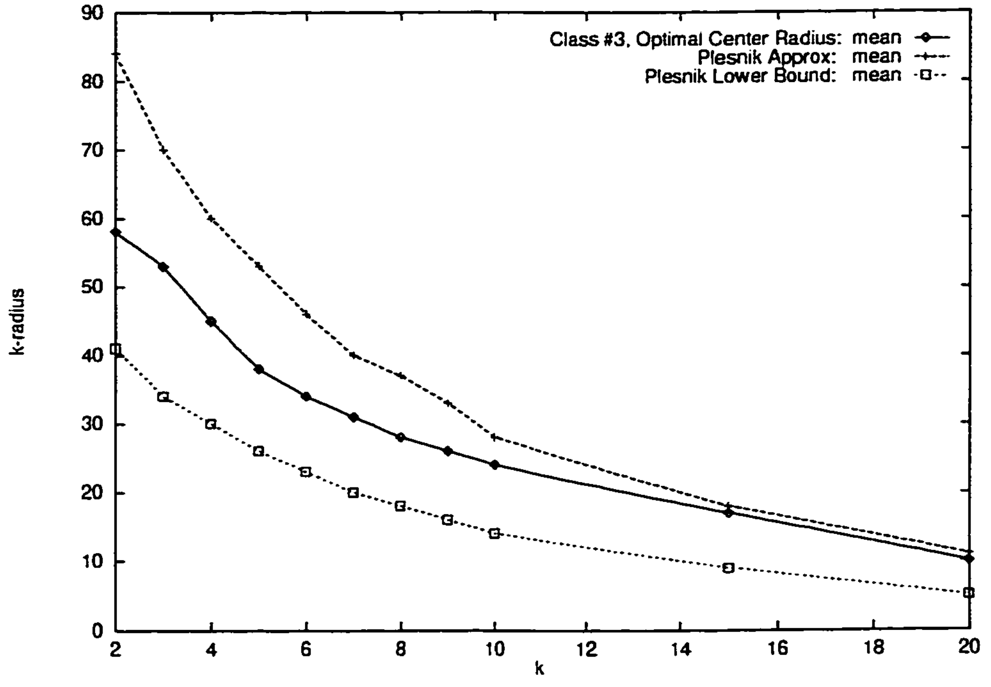

Figure 3.2: Optimal Center Radius vs Approximate Bounds, class #3

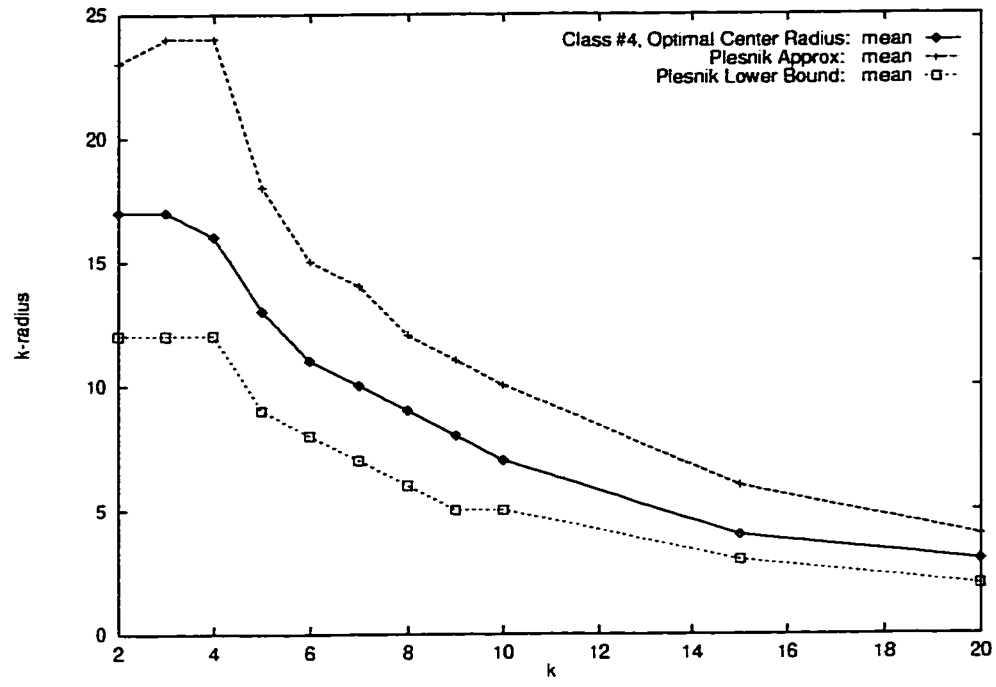

Figure 3.3: Optimal Center Radius vs Approximate Bounds, class #4

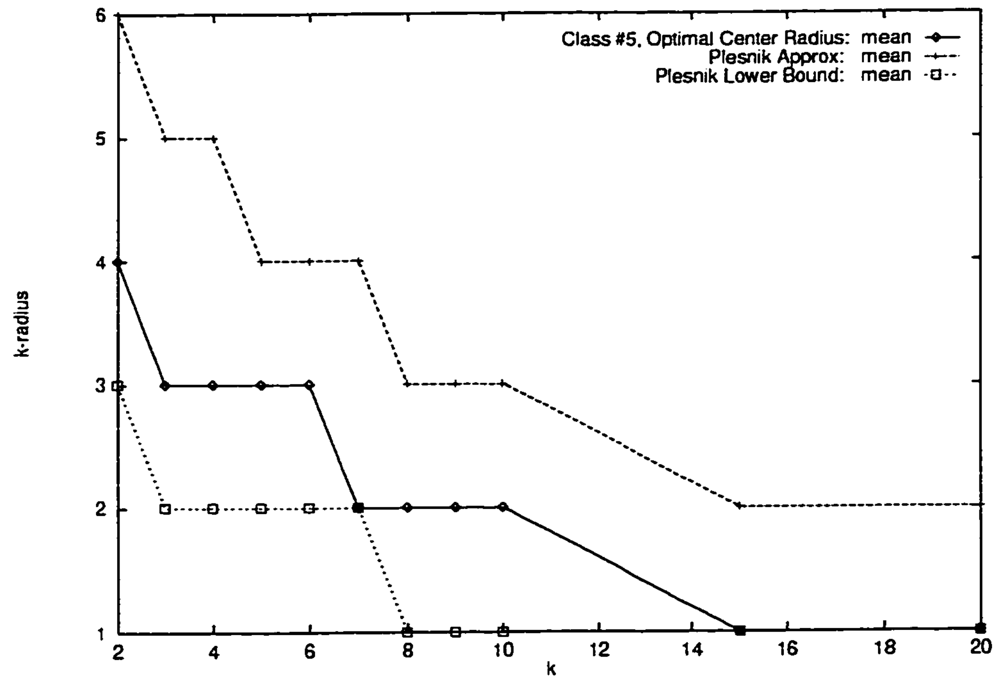

Figure 3.4: Optimal Center Radius vs Approximate Bounds, class #5

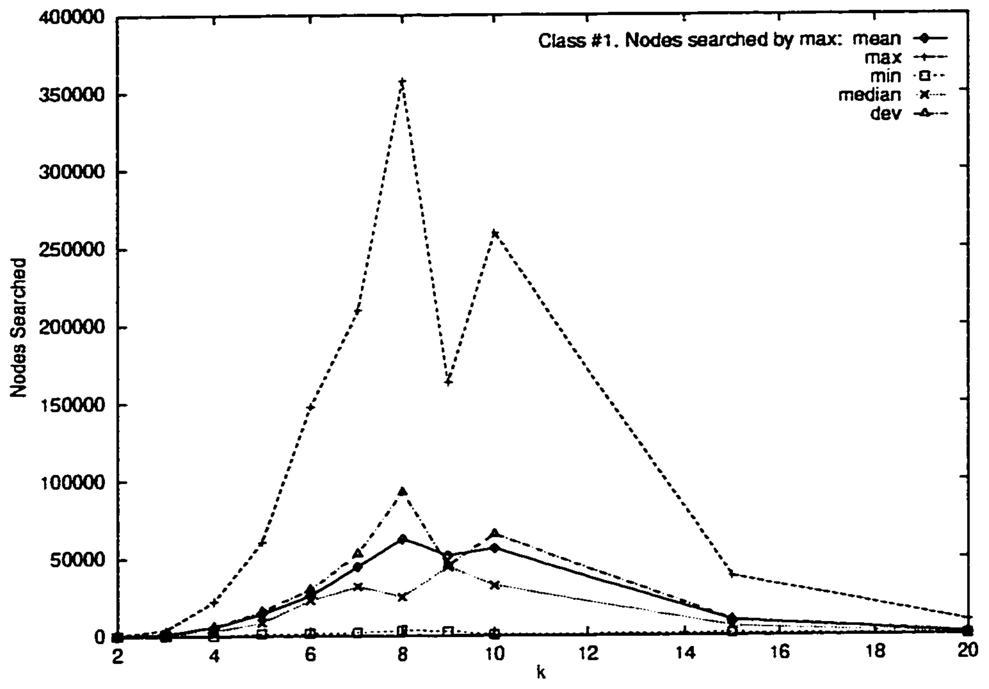

Figure 3.5: Nodes Searched for Optimal solution by Max First/In first on class #1

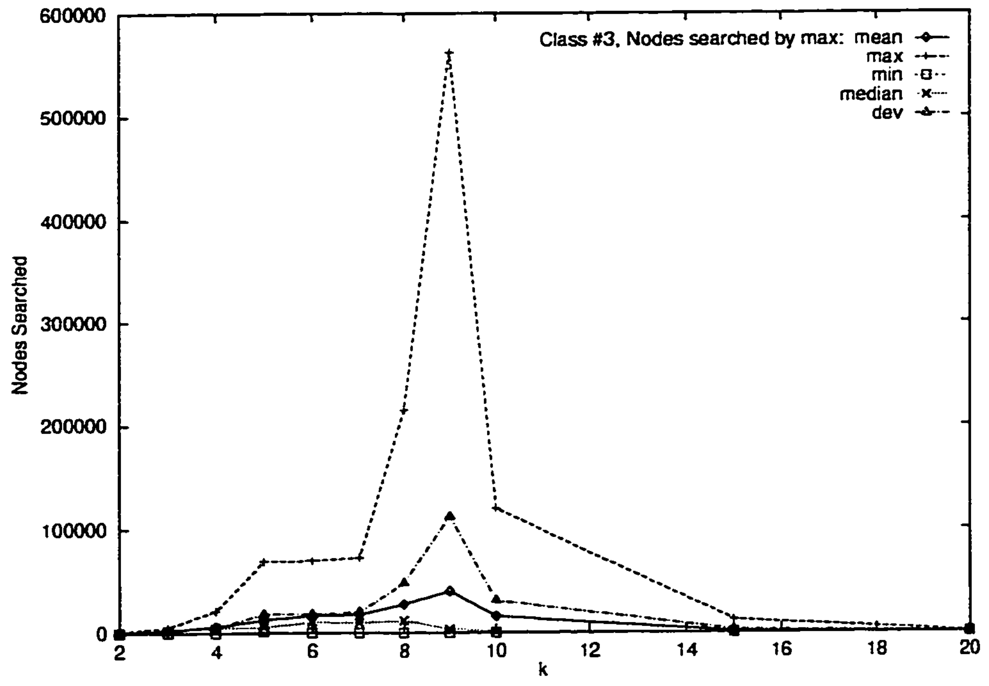

Figure 3.6: Nodes Searched for Optimal solution by Max First/In first on class #3

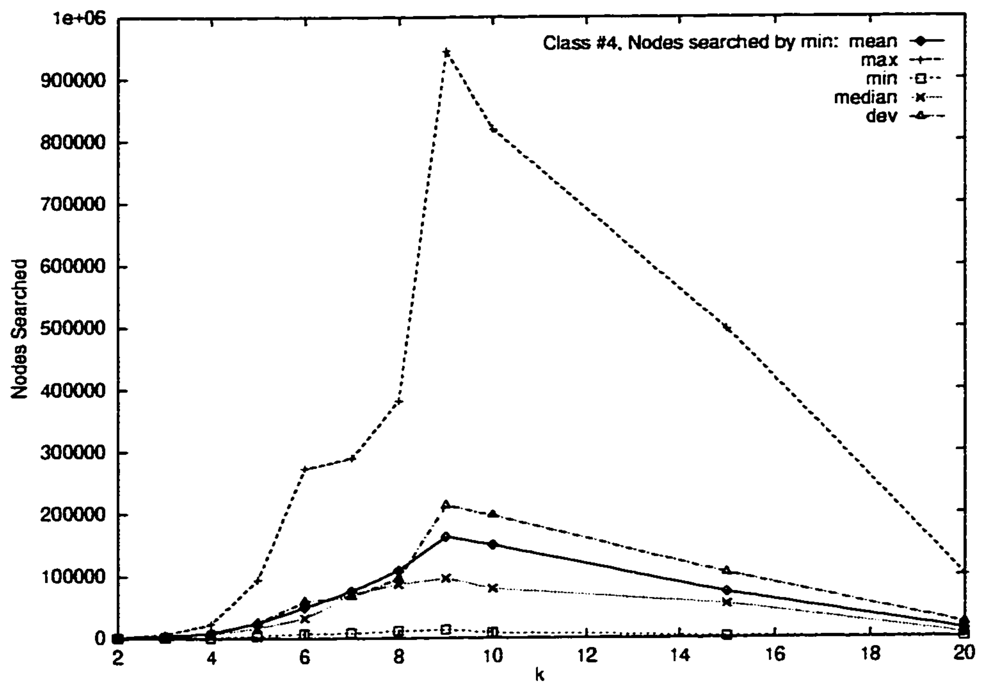

Figure 3.7: Nodes Searched for Optimal solution by Min First/Out first on class #4

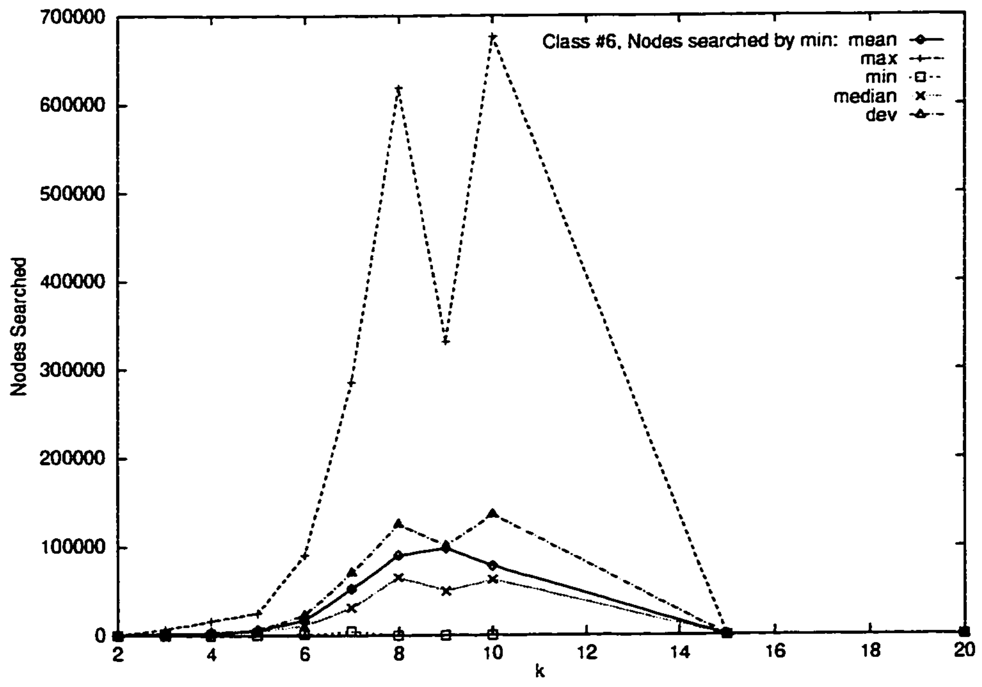

Figure 3.8: Nodes Searched for Optimal solution by Min First/Out first on class #6

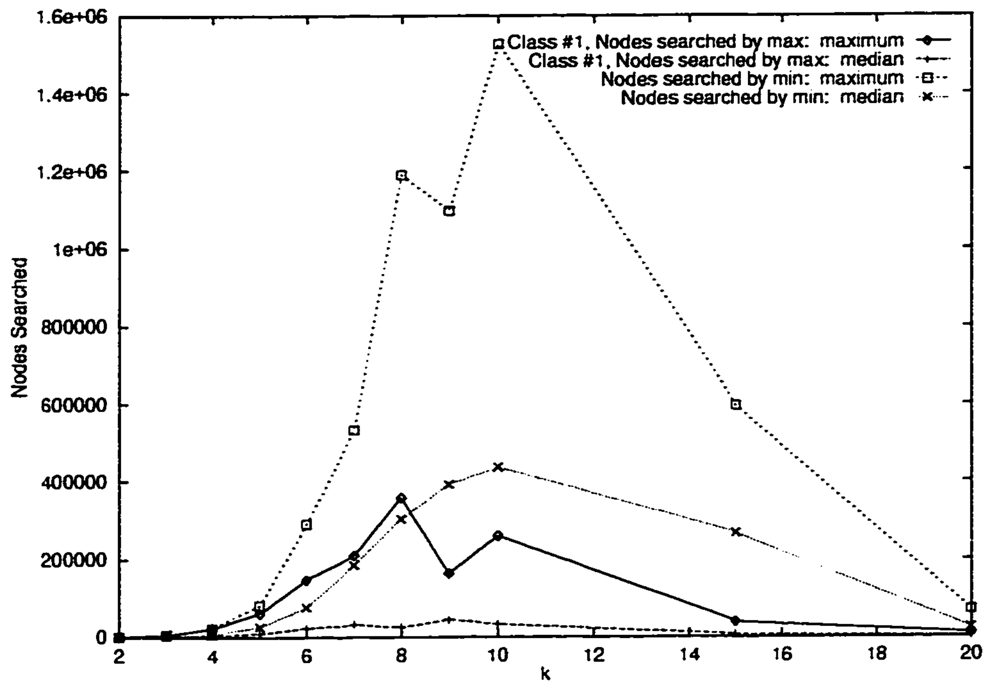

Figure 3.9: Include Max, vs Exclude Min performance in *Depth First Center* on class #1

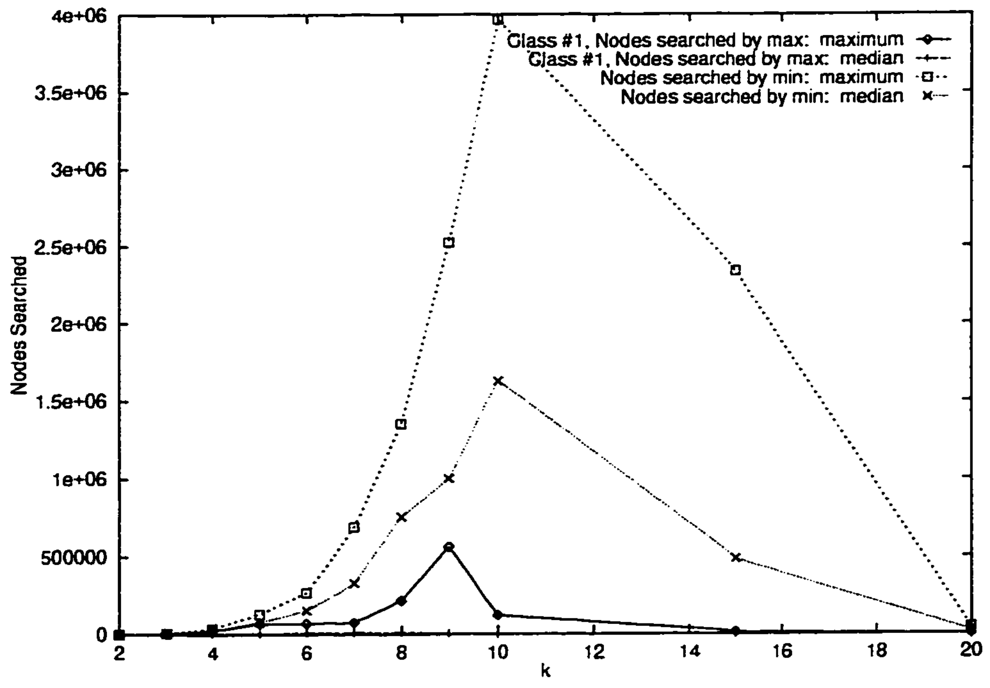

Figure 3.10: Include Max, vs Exclude Min performance in *Depth First Center* on class #3

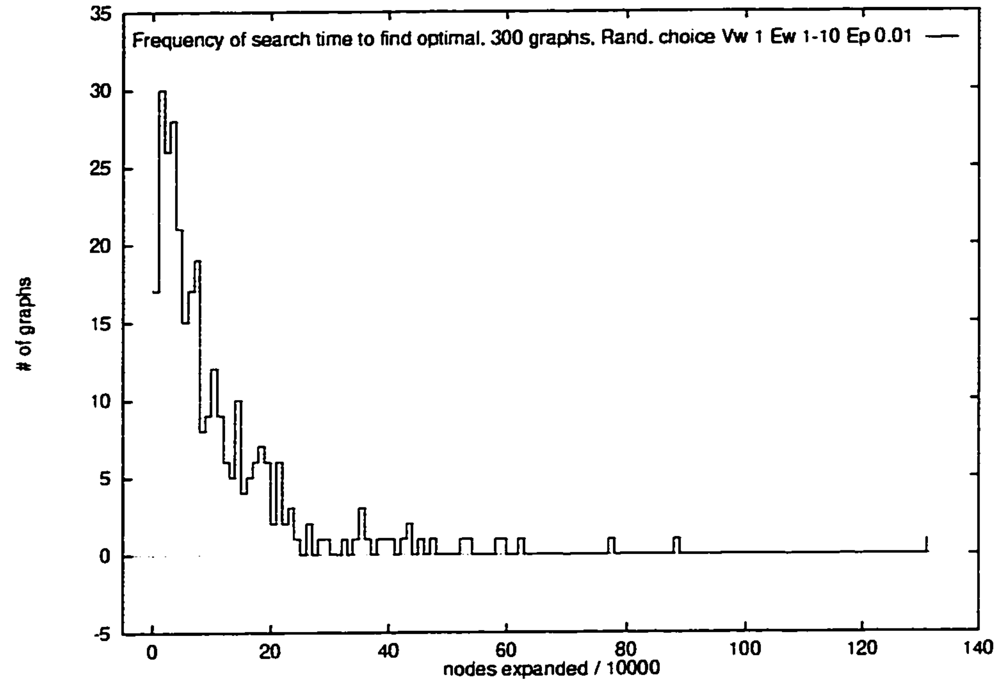

Figure 3.11: Frequency of search times, 300 graphs using *Random Choice / Out First Depth First Center* on class #3

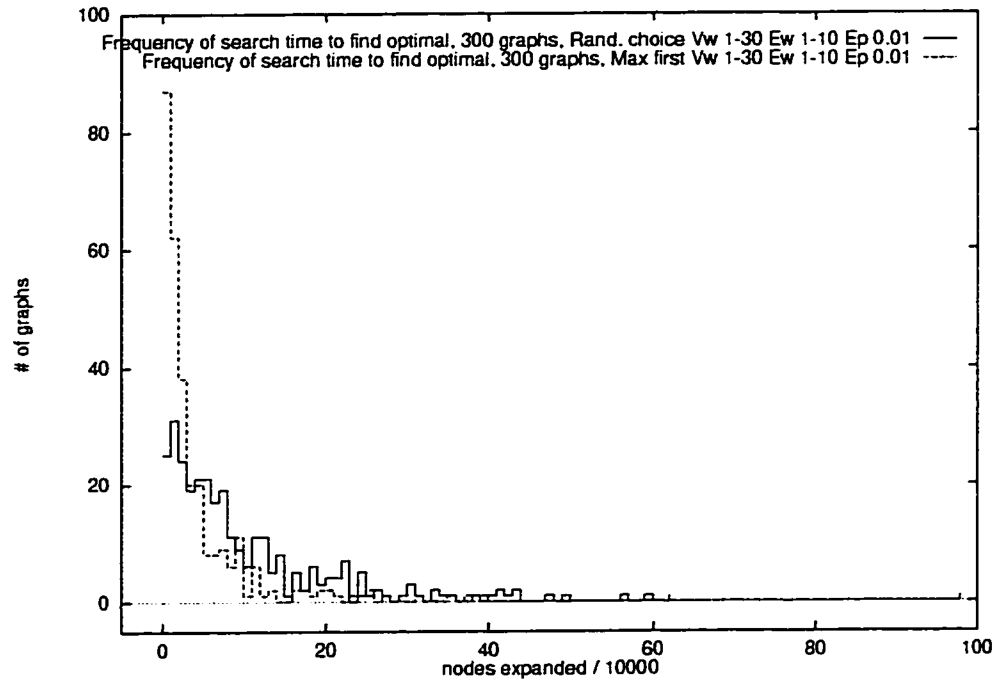

Figure 3.12: Frequency of search times, 300 graphs using *Random Choice / Out First* and *Max First / In First* on class #1

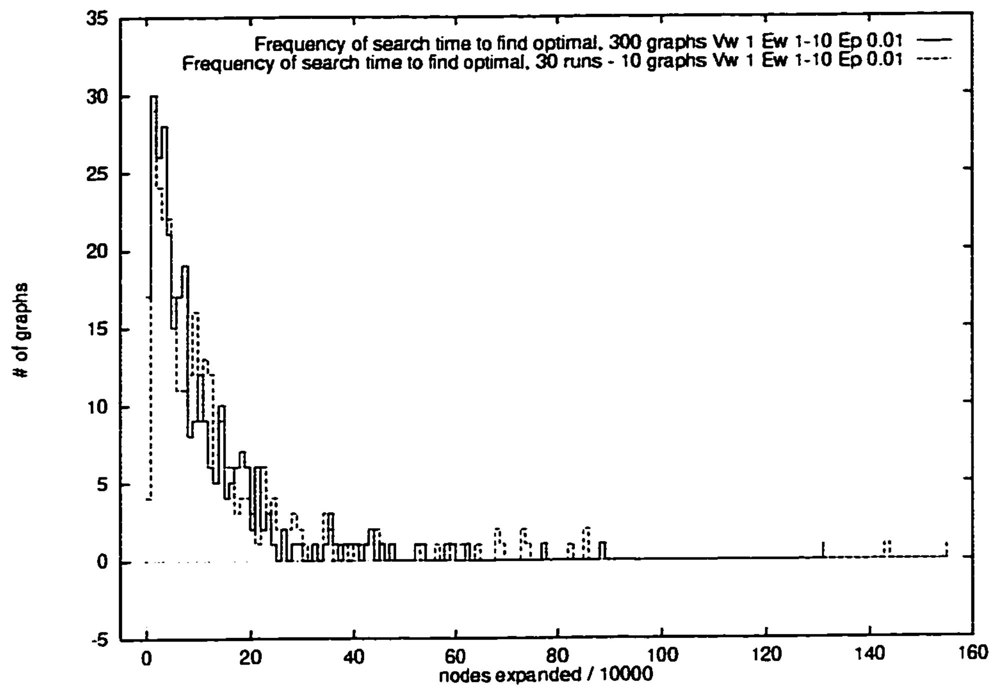

Figure 3.13: Frequency of search times, 1 run on 300 graphs, 30 runs on 10 graphs *Random Choice / Out First Depth First Center* on class #3

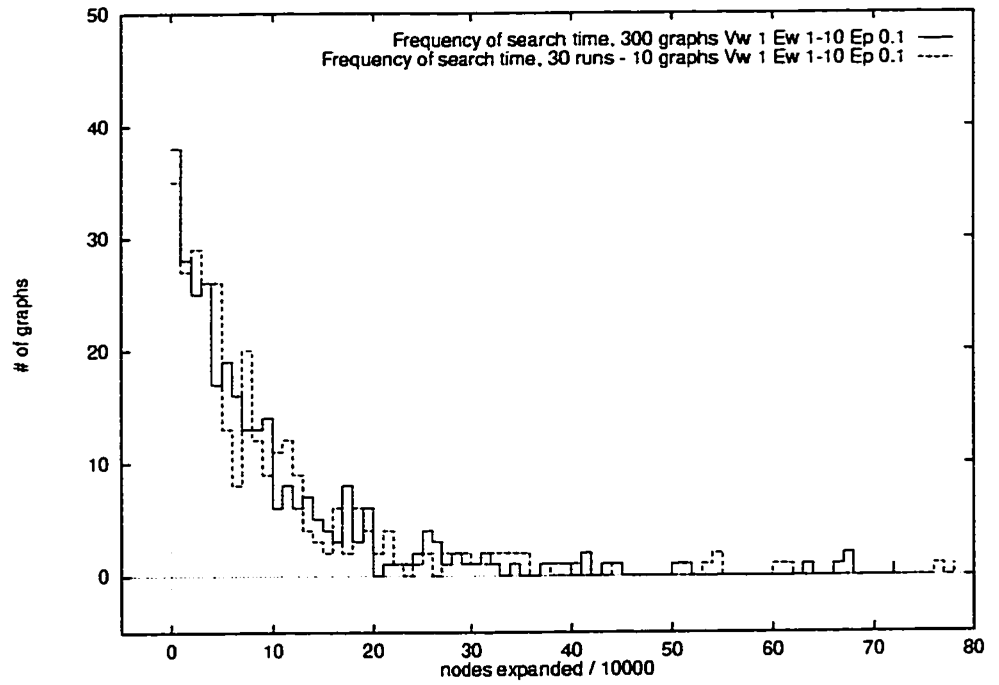

Figure 3.14: Frequency of search times, 1 run on 300 graphs, 30 runs on 10 graphs *Random Choice / Out First Depth First Center* on class #4

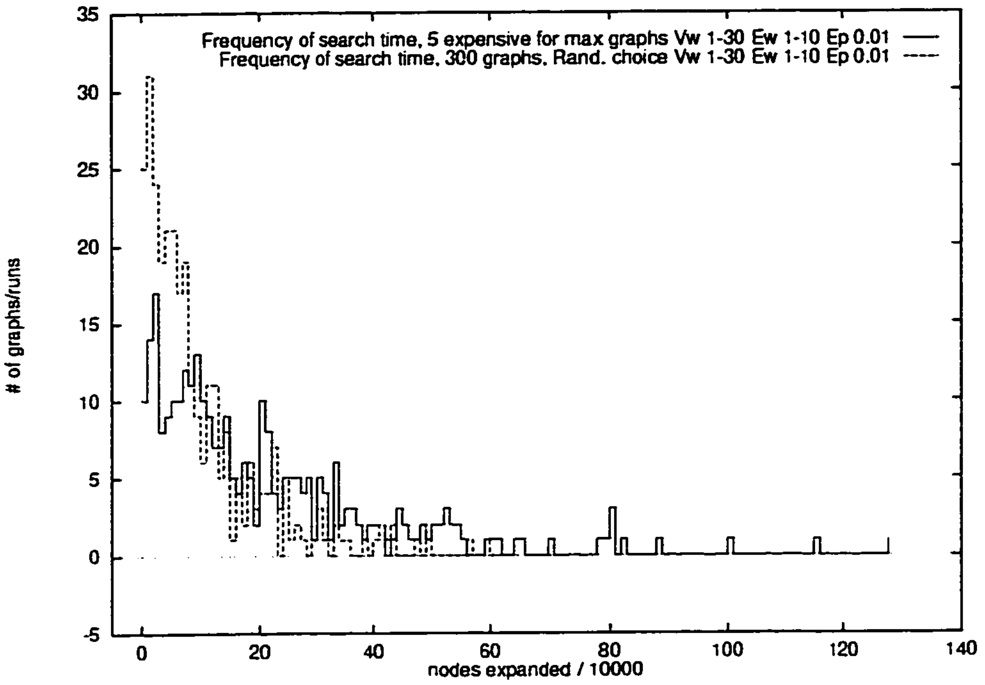

Figure 3.15: Frequency of *Random Choice / Out First* search times, 300 graphs, and 60 runs on 5 expensive graphs for *Max First / In First* from class #1

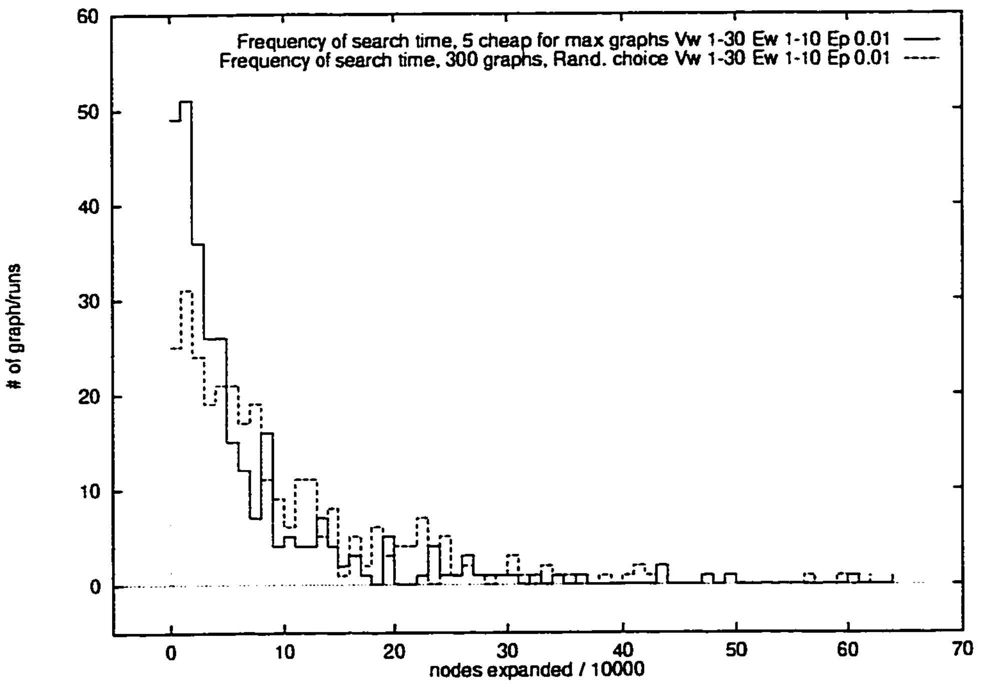

Figure 3.16: Frequency of *Random Choice / Out First* search times, 300 graphs, and 60 runs on 5 inexpensive graphs for *Max First / In First* from class #1

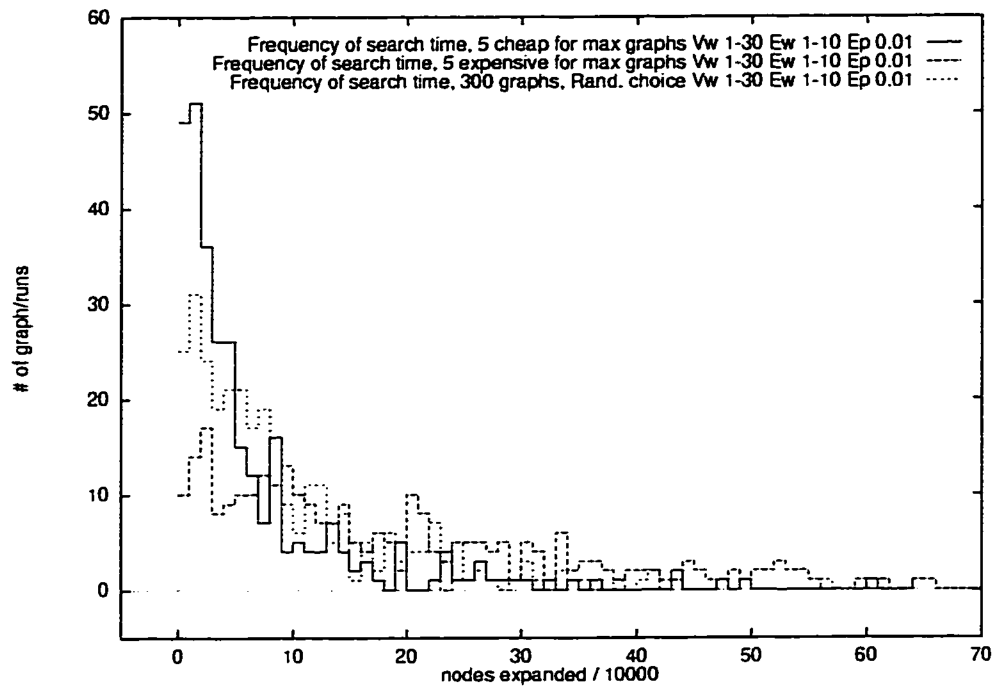

Figure 3.17: Frequency of *Random Choice / Out First* search times, 300 graphs, 60 runs on 5 inexpensive graphs for *Max First / In First* and 60 runs on 5 expensive graphs for *Max First / In First* from class #1

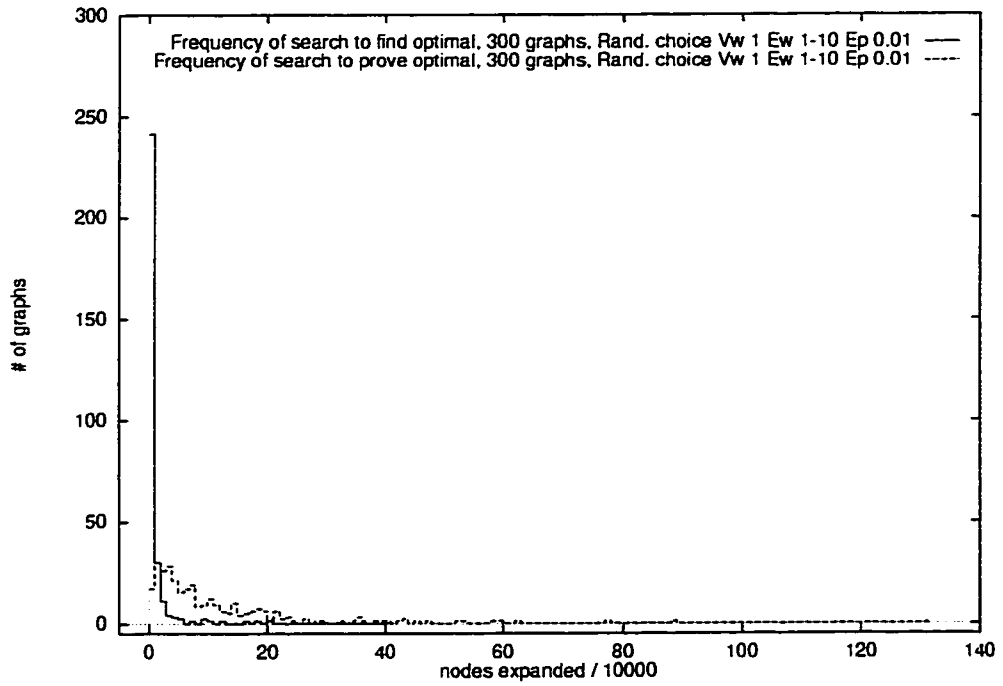

Figure 3.18: Frequency of search times to finding vs. proving optimal: 300 graphs using *Random Choice / Out First Depth First Center* on class #3

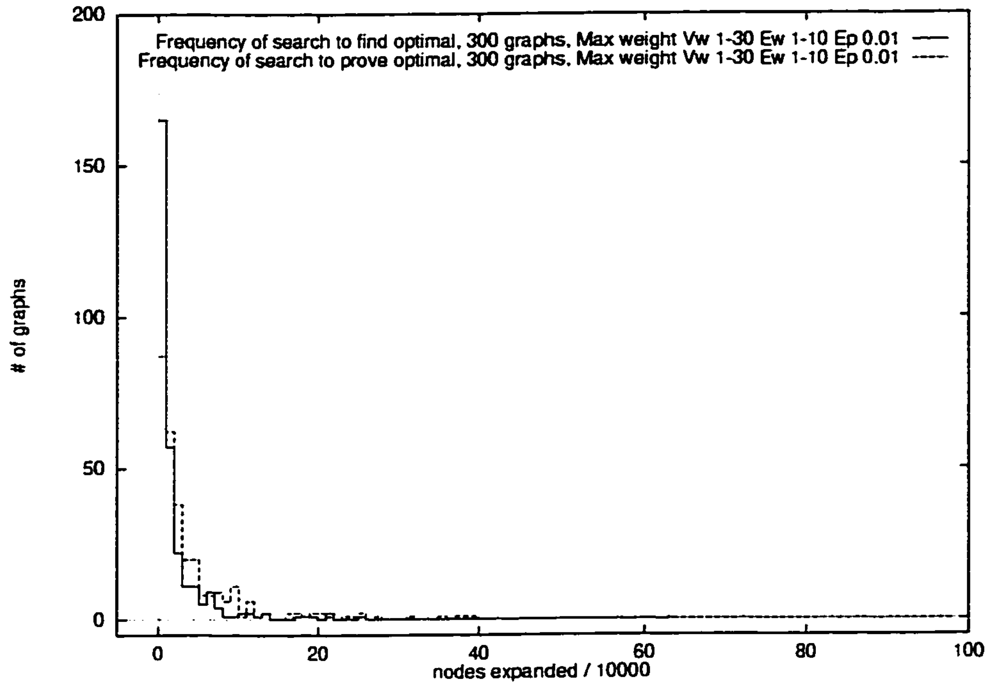

Figure 3.19: Frequency of search times to finding vs. proving optimal, 300 graphs using *Max First / In First* on class #1

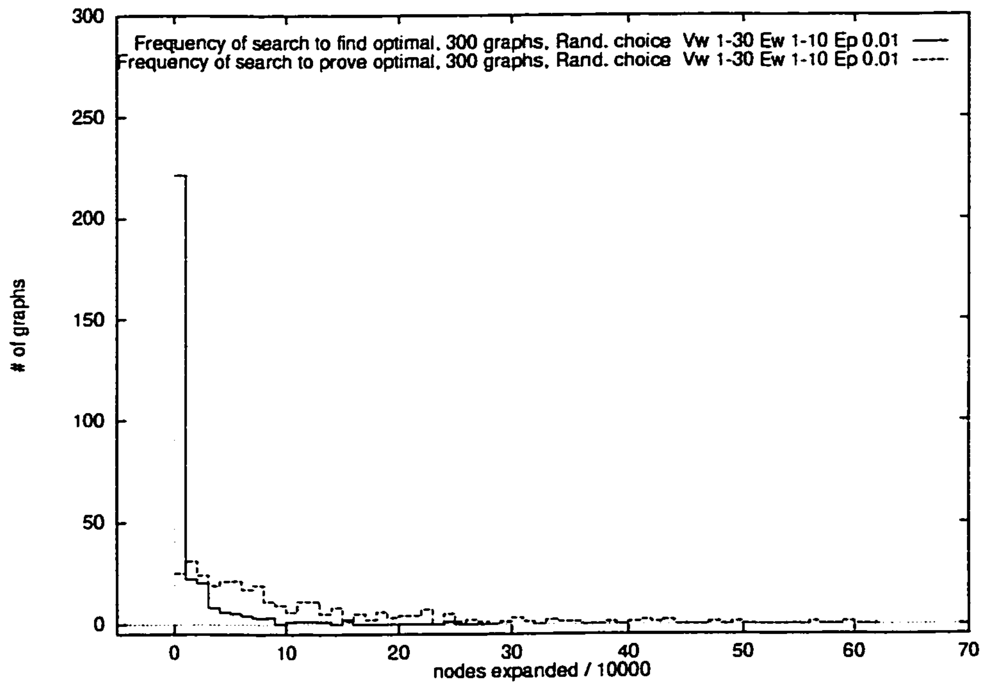

Figure 3.20: Frequency of search times to finding vs. proving optimal, 300 graphs using *Random Choice / Out First* on class #1

# Chapter 4

# Improving Heuristics

## 4.1 Motivation

We know already that the *Plesnik Heuristic Center* algorithm is 2-approximate, and we also know that the 2-approximate bound on a solution is the best absolute bound on a a solution that we can hope for in a polynomial time algorithm unless $P = NP$. Nevertheless, we also recognize that these are worst case bounds and do not tell us how well the algorithm may do in practice. *Plesnik Heuristic Center* is a simple greedy algorithm, with the only heuristic applied being that of selecting a maximum weight vertex in the graph. In this chapter we introduce greedy algorithms similar to *Plesnik Heuristic Center* where the heuristic can take into account other vertices, and attempt to better guide the algorithm's choice of vertices based not only on the weight of the vertex itself, but also on properties of the vertices it may be covering. We will call these heuristics *Ball Heuristics*, and we show that they can be useful in obtaining better solutions in some cases.

## 4.2 *Plesnik Heuristic Center* and Optimality

Procedure RANGE from *Plesnik Heuristic Center* is a greedy algorithm. For some problems, such as graph coloring, we know that a greedy coloring algorithm is guaranteed to find an optimal coloring when the vertices are chosen in a particular order. That is, there exists an ordering of the vertices for which the greedy coloring algorithm will find the optimal coloring.

Procedure RANGE in *Plesnik Heuristic Center* never chooses already covered vertices as members of the set that it is constructing. This is a must for its guarantee of a 2-approximate solution, but also leads to another observation.

**Theorem 9** *There may not be any ordering of vertices for which Plesnik Heuristic Center will find the optimal center.*

The proof is by example. Consider the graph in Figure 4.2

We are looking for a 2-center. The optimal 2-center is $a, b$ with a radius of 2. As the vertices are all of equal weight, procedure RANGE from *Plesnik Heuristic Center* may choose them in any

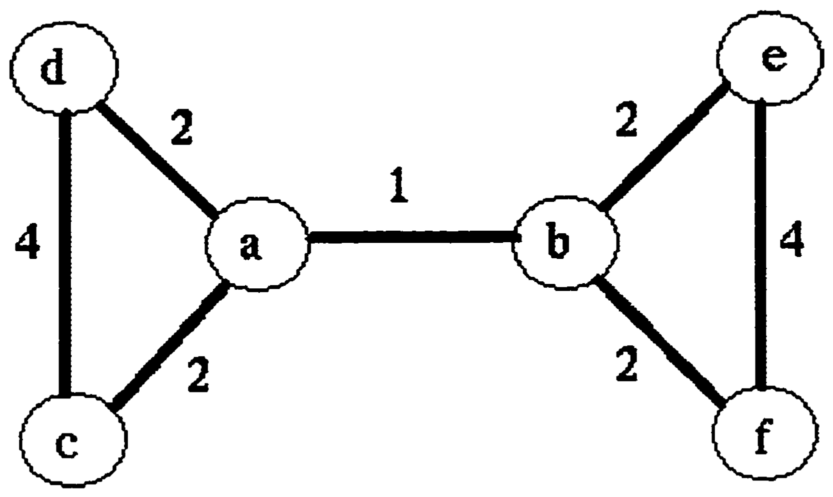

Figure 4.1: A worst case graph for *Plesnik Heuristic Center* finding a 2-center

order. On picking either of $a$ or $b$, when $r=1$, RANGE will cover the other vertex of the optimal center (since $2r = 2$), and be forced to pick a non-optimal vertex. In this case RANGE will be unable to find a 2-center of radius 2, and will find a non-optimal center of radius 4, (exactly twice the optimal). In this particular case, with only two possible choices of center that will produce a radius greater than 4, randomly choosing a 2-center from the graph would be nearly as effective, and would have a chance of finding the optimal (as random choice always will).

We observe that RANGE will never construct a center where two members of the center cover each other. As such it is impossible for RANGE to construct an optimal center in a graph where the only optimal center $X$ has any vertices $x_i, x_j \in X$ with $w(x_j)d(x_i,x_j) \leq \eta(X)$ and $w(x_i)d(x_j,x_i) \leq \eta(X)$.

## 4.3 Fooling the Maximum Weight Heuristic

Both *Plesnik Heuristic Center* and *Max First / In First* depend on a heuristic of choosing a maximum weight vertex first. Knowing this, we can generate a class of graphs on which these algorithms will perform poorly.

We generate a graph $G$ by starting with $n$ vertices. Divide the $n$ vertices into $k$ subsets, $S_1..S_k$ such that $|S_i| \geq 3$. We choose $k$ vertices, $v_1..v_k$, one from each subset $S_1..S_k$, which will be the optimum *k-center*. We assign $v_1..v_k$ each a weight of $w$. We assign all other vertices a weight $w' > w$. We connect each $v_i$ to all other vertices in $S_i$ by an edge of length $l$.

Then we may add new edges arbitrarily to the graph, where any other edges added are of length greater than or equal to $2l$.

Such a graph will have exactly one *k-center* of radius $lw'$, consisting of the vertices $v_1..v_k$, that we initially chose with weights of $w$. All other centers of size $k$ in the graph will have a radius of $2lw'$. In such a case, The *Plesnik Heuristic Center* algorithm will always find an approximate center with the worst-case radius of $2lw'$. This is obvious when we examine procedure RANGE from *Plesnik Heuristic Center.* RANGE always chooses a maximum weight vertex as a member of the approximate center. As such, RANGE will always put at least one vertex of weight $w'$ in the approximate center. Any *k-set* from the graph above containing a vertex of weight $w'$ will always yield a *k-radius* of $2lw'$.

Even in the case where all the other vertices are of weight $w$, *Plesnik Heuristic Center* will not find the optimal unless RANGE is fortunate enough to pick the right vertices by random selection, which for $k$ small relative to $n$ will not be very likely.

These graphs will also be difficult for the *Max First / In First Depth First Center* search algorithm to find an optimal solution. The choice of vertex ordering will ensure that *Max First / In First* makes bad decisions early, only finding the optimal solution at the end of its search after backing out of everything.

## 4.4 Why Maximum Weight is Needed by *Plesnik Heuristic Center*

The *Plesnik Heuristic Center* algorithm relies on choosing a maximum weight vertex in procedure RANGE to guarantee a bound of 2 on the optimal solution. *Plesnik Heuristic Center* searches through all weighted distances in the graph to find the smallest one which, when used as $r$ in procedure RANGE, causes procedure RANGE to return a set of size less than or equal to $k$. The algorithm depends on two conditions to guarantee a 2-approximate solution:

1. The radius of an optimal solution must be one of the weighted distances from the graph.

2. If a *k-center* of radius $\leq r$ exists, then procedure RANGE is guaranteed to find a a set of size $\leq k$ with radius $\leq 2r$.

.

The first condition is true from the definition of the *k-center* problem. If RANGE does not choose a vertex of maximum weight, the second condition will not be met. Consider the proof of Theorem 5 in Chapter 2. If $w(v) > w(u)$ for some vertex $v$ in one of the partitions $S_i$, then the inequalities from the proof will not hold. This means that RANGE might not label some vertices from partition $S_i$ when we choose $u \in S_i$. Once there are vertices in a partition left uncovered after one member is chosen by RANGE, it might have to choose a second vertex from the partition, and would not be guaranteed to output a set of size $\leq 2r$ if there is a *k-set* $X$ with $\eta(X) \leq r$.

## 4.5 Modifying *Plesnik Heuristic Center*

.

In order to guarantee a 2-approximate solution, any other heuristic we use in *Plesnik Heuristic Center* must guarantee the choosing of a vertex of weight high enough so that we can guarantee that the inequalities of Theorem 5 will hold. For Therotem 5 to hold it is sufficient that the vertex chosen at each step in RANGE be of greater weight than any vertex within *distance* (not weighted distance) $2r$ from itself. If this is the case we can be assured that any vertex in the set $S_i$ will be of lesser weight than the chosen vertex, and Theorem 5 will hold. This observation is of possible use as it tells us that we may be able to pick a vertex that is not a maximum over the whole graph and still do well, particularly where large weight vertices are separated by large distances.

Of course, in a vertex-unweighted graph (or any graph where all vertex weights are equal), all vertices are of maximal weight. We may therefore use a heuristic that chooses them in any order, and we will still obtain a 2-approximate solution.

In general, if RANGE were not to pick a maximum weight vertex, the guarantee on the approximate value of the solution would change. Specifically, if the vertex chosen could differ from the maximum by a factor of $\zeta \geq 1$, i.e. $\zeta w(u) \geq w(v)$ for all $v \in V(G)$, then the inequalities from the proof of Theorem 5 change to be $w(v)d(u,v) \leq w(v)[d(u,x_i) + d(x_i,v)] \leq \zeta.2r$, and $w(u)d(x_i,u) + w(v)d(x_i,v) \leq \zeta.2r$. This will imply that we would then need to modify RANGE to cover everything within $\zeta.2r$ in order to guarantee that it finds a set of size $\leq k$ when a *k-set* of radius $r$ exists. Using such a modification would then give a provably $\zeta 2$-appoximate solution.

## 4.6 Ball Heuristics

The heuristic used for vertex ordering in both *Plesnik Heuristic Center* and *Max First / In First*is that of maximum vertex weight. In *Plesnik Heuristic Center*, this guarantees us a 2-approximate solution. It is generally desirable to have large weight vertices in the center, as a high weight vertex in a center is covered "for free". On the other hand, if we need to cover it from another vertex, its large weight makes a large weighted distance that possibly gives us a large radius for our solution. In a vertex weighted graph, it is useful to cover the large weight vertices in short distances to keep the radius low.

With this in mind we consider extending the maximum vertex weight heuristic to encompass the idea not only of how much weight we cover "for free", but how much weight we can cover "inexpensively". For some vertex $u$, we consider all vertices within some weighted distance $r$, $X = \{v \in V | w(v)d(u,v) < r\}$, and we assign a value $h(u) = f(u,r)$. We will call this sort of heuristic, where we consider "balls" of nearby vertices *Ball Heuristics*.

Note that choosing a maximum weight vertex corresponds to a Ball Heuristic where the weighted distance used to determine the size of the ball is 0. We are therefore examining what happens when we expand the ball of vertices used to evaluate a vertex beyond a weighted distance of 0.

Consider the following two candidates:

**Definition 19** *First, let $B(r,u) = \{v \in V | w(v)d(u,v) \leq r\}$. We call $B(r,u)$ a ball of radius $r$ around $u$.*

**Definition 20**

*The total weight ball heuristic, for radius $r$, vertex $u$, is the sum of the weights of all vertices $v$ in the ball of radius $r$ around $u$, i.e. $T(u,r) = \sum_{v \in B(r,u)} w(v)$*

**Definition 21**

*The mean weight ball heuristic, for radius $r$, vertex $u$, is the mean weight of all vertices $v$ in the ball of radius $r$ around $u$, i.e. $M(u,r) = mean_{\{v \in B(r,u)\}} w(v)$*

When the radius is 0 both these heuristics are exactly equivalent to choosing a max weight vertex. For positive values of $r$, they may assign different weightings to vertices. The total weight ball heuristic simply measures the total amount of vertex weight within our ball radius. The mean weight ball heuristic will favor small numbers of larger weight vertices within the radius, as opposed to large numbers of small weight vertices. If we compare the case of a large number of small vertices close to our candidate, against the case of a small number of large vertices nearby, we probably want to choose the one with the large vertices nearby, for the reason that we can hope to cover the small nearby vertices from other places inexpensively. Given two vertices with the same total ball heuristic value, and in absence of other criteria, it would therefore appear to be desirable to choose the one with the higher mean ball heuristic value. We will examine the use of the total weight ball heuristic, with the mean weight ball heuristic used to break ties. Vertices of equal total ball weight will be differentiated by the mean ball weight, with the higher mean ball weight being ranked as a better choice by the heuristic than the smaller mean.

If we wish to use a ball heuristic in the *Depth First Center* algorithm we must choose a suitable value of $r$ for which to compute the heuristic and order the vertices for search. A promising candidate for $r$ might be the lower bound from an application of the *Plesnik Heuristic Center* algorithm. Alternatively, in practice it may suffice to use a value for $r$ chosen to be "appropriately" smaller than center values found by some other quick approximation method such as *Dyer and Freize Heuristic Center* or a few random samples.

Given a distance graph $D(G)$, the value of ***total weight ball heuristic*** and ***mean weight ball heuristic*** can be computed for all vertices in the graph in $O(n^2)$ time. For *Plesnik Heuristic Center*, in procedure RANGE, this heuristic could be computed for each value of $r$ tried, as there will be $O(log(n))$ values of $r$ tried by *Plesnik Heuristic Center.* Thus the complexity of using the heuristic for all values of $r$ tried in *Plesnik Heuristic Center* is $O(n^2 log(n))$, and doing so would not increase the worst case complexity of the algorithm by more than a constant factor. We also know that using any strategy in RANGE for vertex selection that does not guarantee a maximum weight vertex being selected will mean we do not get a provably 2-approximate *k-center* from *Plesnik Heuristic Center.*

Our goal in using the heuristic is to choose good vertices based on how much weight (including themselves) can be covered "inexpensively", in the hope of finding a good center. So it may be a good idea to establish a bound on the solution by using the conventional *Plesnik Heuristic Center* algorithm first. Once a lower and upper bound on the optimal solution has been established by this search, we then search using the ball heuristic computed at some radius (perhaps the lower bound) in the hope of finding a better center. This would ensure that we obtain a 2-approximate center due to the initial bounds established by the *Plesnik Heuristic Center* algorithm.

Another way to use a ball heuristic would be to use it as a secondary selection strategy in RANGE, i.e. Choose a maximum weight vertex of the highest ball heuristic value. When used in this manner, we would still guarantee a 2-approximate solution. In the case where there are multiple vertices of maximum weight, or when the graph is vertex unweighted, we hope to get a better center

A slightly different approach could be taken to improve the lower bound on the optimal solution. If we use a strategy of choosing a maximum weight vertex with the *lowest* ball heuristic value, we are still guaranteed a 2-approximate solution, but we are now attempting to find the worst 2-approximate solution we can to make the lower bound as high as possible. Given another (hopefully better) approximate solution a high lower bound could increase our confidence in how good the approximate solution is.

As the ball heuristics could assign high values to low weight vertices, using only a ball heuristic in the vertex selection strategy of RANGE will mean we are not guaranteed a 2-approximate solution. However, as we have shown previously, we can have optimal centers in a graph that do not contain vertices of maximum weight. While we know that using such a heuristic in RANGE may result in a poor solution, we do not yet know how good a solution this can produce in most situations. It may well be that using a ball heuristic to choose vertices in a greedy algorithm similar to *Plesnik Heuristic Center* will give us better centers for some or many classes of graphs, in spite of a poorer bound. We may want to consider the use of such a method in an attempt to find a better approximate center on average than is found by the conventional *Plesnik Heuristic Center* algorithm.

A potential problem with using a ball heuristic may be that the computed values will become less and less useful as vertices are chosen. Consider the case of constructing an approximate center by choosing vertices sorted by an initially computed ball heuristic value in the same manner as RANGE. We label vertices we consider as covered by each vertex as we add it to the center. As more and more vertices are covered, it is more and more likely that they are vertices whose weights contributed to the ball values of the remaining uncovered vertices. Without recomputing the ball heuristic values during the course of picking the vertices, the high heuristic values may not represent a good ranking of the remaining vertices. One alternative is to recompute the heuristic values, taking into account only the uncovered vertices, at each step after a vertex is chosen. Another alternative is to pick a point at which to abandon the use of, or assign less and less value to the previously computed ball value of a vertex (possibly reverting to choosing by max weight alone at some point in constructing a center).

If we are going to consider any algorithm where we do not have a guaranteed 2-approximate solution, it may be worthwhile for us to modify RANGE to choose an already covered vertex as a member of the set it is constructing if it appears advantageous to do so. RANGE labels vertices as covered by the members of the set it constructs, and it will never choose a covered vertex as a member of the set. As we have seen, this property can be used by an adversary to guarantee that *Plesnik Heuristic Center* will always obtain a worst-case solution. If we are using a ball heuristic, we may recompute the ball heuristic values after the addition of each vertex to our approximate center. If we consider only uncovered vertices in the recomputed ball heuristic value, a high heuristic value for a vertex that has been covered by a previous vertex can indicate that it might still be a good choice to add to the set. The intent of this modification is to construct a greedy heuristic for centering that will be able to pick centers where members of a center are within covering distance of each other.

We then have a number of options to consider in how we can apply a ball heuristic in a similar algorithm to *Plesnik Heuristic Center.*

- As a secondary selection strategy for maximum weight vertices, to either attempt to get a better approximate solution, or a higher lower bound.
- Computed at some fixed radius, then used as a selection strategy to greedily attempt to find a good center.
- Recomputed after each vertex added, possibly allowing previously "covered" vertices to be chosen.

We consider the following algorithms:

**Secondary Ball Heuristic Center**

Input: Weighted graph $G$, integer $k$, ball heuristic $B$, ball radius $b$,

Output: *k-setS*, and lower bound $r$.

**Step 1** Construct if not given, and sort the $n(n-1)$-multi-set of weighted distances $d(u,v)(w(v)$ for $u, v \in V$ into a non-decreasing sequence. Delete duplicates to reduce it to an increasing sequence $\{f_1 < f_2 < \cdots < f_q\}$ .

**Step 2** Find $r'$, the least value of $r \in \{f_1 < f_2 < \cdots < f_q\}$ for which GREED yields an output $S'$ with $|S'| \leq k$, using $H'$, and ordering of vertices by maximum weight first.

**Step 3** Find $r'' \in \{f_1 < f_2 < \cdots <= 2r'\}$, the least value of $r$ for which GREED yields an output $S''$ with $|S''| \leq k$, using $H''$, an ordering of the vertices by maximum weight and highest value of ball heuristic B at computed at radius $b$.

**Step 4** Find $r''' \in \{f_1 < f_2 < \cdots <= 2r'\}$, the least value of $r$ for which GREED yields an output $S'''$ with $|S'''| \leq k$, using $H'''$, an ordering of the vertices by maximum weight and lowest value of ball heuristic $B$ computed at radius $b$.

**Step 5** Augment each of $S'$, $S''$, and $S'''$ to a set of $k$ vertices by choosing vertices of high vertex weight, and high value of ball heuristic $B$

**Step 6** Output the lesser radius set from $S'$, $S''$, and $S'''$ as $S$, and output $max(r', r'', r''')$ as $r$, stop.

**Procedure GREED**

: Input: Weighted graph $G$, integer $k$, heuristic ordering of vertices $H$.

: Output: Set $S$, a center of radius $\leq 2r$.

**Step 0** At first, all vertices of $G$ are unlabeled; $S \leftarrow \phi$.

**Step 1** While there are unlabeled vertices, choose an unlabeled vertex $u$ of maximum heuristic value by ordering $H$, and put $S \leftarrow S \cup \{u\}$; label the vertex $u$ and every unlabeled vertex $v$ such that $w(v)d(u,v) \leq 2r$.

**Step 2** Output $S$.

**Simple Ball Heuristic Center**

Input: Weighted graph $G$, integer $k$, ball heuristic $B$. Distance max $D$, ball radius $b$.

Output: *k-setS*.

**Step 1** Construct if not given, and sort the $n(n-1)$-multi-set of weighted distances $d(u,v)(w(v)$ for $u, v \in V$ into a non-decreasing sequence. Delete duplicates to reduce it to an increasing sequence $\{f_1 < f_2 < \cdots < f_q\}$ .

**Step 2** Find $r' \in \{f_1 < f_2 < \cdots \leq f_q\}$, the least value of $r$ for which GREED yields an output $S'$ with $|S'| \leq k$, using $H$, an ordering of the vertices by the highest value of ball heuristic B at computed at radius $b$.

**Step 3** augment $S'$ to a set $S''$ of $k$ vertices by choosing vertices of highest value of $B$ that are not already in $S'$. Output $S''$ and stop

**Procedure GREED**

: Input: Weighted graph $G$, integer $k$, heuristic ordering of vertices $H$.

: Output: Set $S$, a center of radius $\leq 2r$.

**Step 0** At first, all vertices of $G$ are unlabeled: $S \leftarrow \phi$.

**Step 1** While there are unlabeled vertices, choose an unlabeled vertex $u$ of maximum heuristic value by ordering $H$, and put $S \leftarrow S \cup \{u\}$; label the vertex $u$ and every unlabeled vertex $v$ such that $w(v)d(u,v) \leq 2r$.

**Step 2** Output $S$.

**Recomputed Ball Heuristic Center**

Input: Weighted graph $G$, integer $k$, ball heuristic $B$. Distance range $D_l \cdots D_h$, ball radius $b$.

**Step 1** Construct if not given, and sort the $n(n-1)$-multi-set of weighted distances $d(u, v)(w(v)$ for $u, v \in V$ into a non-decreasing sequence. Delete duplicates to reduce it to an increasing sequence $\{f_1 < f_2 < \cdots < f_q\}$ .

**Step 2** Find $r'$, the least value of $r \in \{D_l \cdots Dh\}$ for which BIG GREED using $B$ computed at radius $b$ yields an output $S'$ with $|S'| \leq k$

**Step 4** augment $S'$ to a set $S''$ of $k$ vertices by choosing vertices of highest value of $B$ that are not already in $S'$. Output $S''$ and stop

**Procedure BIG GREED**

: Input: Weighted graph $G$, integer $k$, ball heuristic $B$, distances $b$, $r$.

: Output: Set $S$, a center of radius $\leq 2r$

**Step 0** At first, all vertices of $G$ are unlabeled: $S \leftarrow \phi$.

**Step 1** While there are unlabeled vertices, compute the value of $B$ at radius $b$ for all vertices (including the labeled ones), counting only unlabeled vertices to the value of $B$. Choose a vertex $u$ of maximum value of $B$. $S \leftarrow S \cup \{u\}$; label the vertex $u$ and every unlabeled vertex $v$ such that $w(v)d(u, v) \leq 2r$.

**Step 2** Output $S$.

Secondary Ball Heuristic Center is a 2-approximate solution to *k-center*. The proof is identical to the proof that *Plesnik Heuristic Center* is 2-approximate. Secondary Ball Heuristic Center differs in that it uses a Ball heuristic to differentiate between vertices of the same weight, attempting to pick vertices for a small center in one pass, and then attempting to pick vertices for a large lower bound in another. In this way, Secondary Ball Heuristic Center attempts to find the narrowest bounds it can on the optimal solution. We expect Secondary Ball Heuristic Center to be an improvement over *Plesnik Heuristic Center* on vertex unweighted graphs, or graphs in which there are numbers of vertices of similar weight. It will of course be no different from *Plesnik Heuristic Center* if no vertices share the same weight.

Simple Ball Heuristic Center is not guaranteed 2-approximate on vertex weighted graphs. It is 2-approximate on vertex unweighted graphs (as every vertex we choose is of maximum weight, and so the proof as for *Plesnik Heuristic Center* holds). We anticipate that this algorithm may not do as well on larger size centers, since it would be more likely to have vertices in a center contributing to a the heuristic value of a vertex, making it appear to be a better choice than it really is. We anticipate that it should do well on vertex unweighted graphs, and for small center sizes.

Recomputed Ball Heuristic Center is not guaranteed 2-approximate on vertex weighted graphs. It does recompute the heuristic as it chooses vertices in BIGGREED, thereby keeping them representative of the vertices remaining to be covered at any step. Due to it being allowed to choose labeled vertices, it won't be provably 2-approximate for vertex unweighted graphs either. Nevertheless, it is intended to try to greedily obtain a better center than obtainable by the other algorithms in many cases. It does this at some additional cost. Due to the recomputation of the Ball heuristic O(n) times, Recomputed Ball Heuristic Center will be $O(n^3 log(n))$. This algorithm is really intended to be run to obtain a better center after a bound has been established using another algorithm such as *Plesnik Heuristic Center*.

# Chapter 5

# Experimental Evaluation of Ball Heuristics

As we have noted, we expect certain properties to affect the success of our Ball heuristic algorithms at finding good approximate solutions to the *k-center* problem.

1. Secondary Ball Heuristic Center should give better approximations and a larger lower bound than *Plesnik Heuristic Center* on vertex unweighted graphs, or graphs with vertices of the same weight.

2. Simple Ball Heuristic Center may give better approximations than *Plesnik Heuristic Center* for small center sizes. It may do worse for large center sizes.

3. Recomputed Ball Heuristic Center should give better approximations on average than *Plesnik Heuristic Center*, when used to improve the approximate center after a run of *Plesnik Heuristic Center*

4. Overall, we expect any improvement by the Ball Heuristic algorithms over *Plesnik Heuristic Center* to be more pronounced on graphs with fewer distinct vertex weights

.

We base these hypotheses on our previous observations of the cases that will induce poor behavior when *Plesnik Heuristic Center* chooses vertices by maximum weight alone, and our observation of the behavior of *Plesnik Heuristic Center* on vertex unweighted graphs in our previous experiments.

To test these hypothesis we implemented and tested each algorithm against random graphs generated to vary the number of distinct vertex weights in the graph. We recorded the solutions obtained by each algorithm for each random graph, and measured the differences in the solutions obtained by each algorithm for each graph. For all the ball heuristic algorithms, the ball was computed using the value of the lower bound from *Plesnik Heuristic Center* as the radius $b$ where the ball heuristic is computed.

We will chose to run our algorithms against graph classes as follows:

1. $\mathcal{G}_{100,[1..20],[1..10],[uniform\frac{1}{20}],[uniform\frac{1}{100}]}$

2. $\mathcal{G}_{100,[1..5],[1..10],[uniform\frac{1}{5}],[uniform\frac{1}{100}]}$

3. $\mathcal{G}_{100,[1],[1..10],[1],[uniform\frac{1}{100}]}$

4. $\mathcal{G}_{100,[1..20],[1..50],[uniform\frac{1}{20}],[uniform\frac{1}{500}]}$

5. $\mathcal{G}_{100,[1..5],[1..50],[uniform\frac{1}{5}],[uniform\frac{1}{500}]}$

6. $\mathcal{G}_{100,[1],[1..50],[1],[uniform\frac{1}{500}]}$

These classes are chosen to be of a size which we test quickly enough to run a sizable sample, and to be representative of varying the vertex weights, edge weights and edge probability.

We will test each algorithm against 20 graphs from each class. On each graph, we will compute approximate centers of size 7, 21, 35, and 49, chosen to be spaced evenly over the interval $[0..\frac{n}{2}]$. We choose to test sizes of $\frac{k}{n} \leq 1/2$, as we noted previously that the problem becomes easier to solve exactly for larger values of $k$.

In order to measure relative performance of the approximation algorithms on randomly generated instances we will compare them on a per-graph basis using the *Plesnik Heuristic Center* algorithm's approximation and lower bound as a basis for the comparison. We will also compare the *Dyer and Freize Heuristic Center* algorithm and an algorithm of repeated random trials as below, where we limit the computation to an amount similar to that used for the most expensive of the algorithms tried.

**Random K-center**

Input: Weighted graph $G$, integer $k$, Integer $T$.

Output: A center of size $k$.

**Repeat** Step 1 and Step 2 $T$ times:

**Step 1** Choose a set $X$ of $k$ vertices randomly from G as a center.

**Step 2** Compute the radius of center $X$ in $G$, compare it to the radius of set $BEST$, the lowest seen so far. if $X$ yields a lower radius, set $BEST$ to be $X$.

output $BEST$,

By comparing each algorithm based upon their relative performance on each random problem instance, we hope to reduce the variance in the resulting data we see in the same manner as described in [12]. In this manner we can measure the relative performance of all the algorithms to the *Plesnik Heuristic Center* approximation algorithm.

## 5.1 Results

In summary, the results were as follows:

- Recomputed Ball Heuristic Center did not behave as we had hoped, and in fact normally performed much worse than any other algorithm except for random search.
- Secondary Ball Heuristic Center behaved as we predicted. It gave better approximations and a larger lower bound than did *Plesnik Heuristic Center*. The difference in approximations and lower bound was much more pronounced on vertex unweighted graphs or graphs with many vertices of the same weight.
- Simple Ball Heuristic Center also behaved as we predicted, giving better approximations for small center sizes than *Plesnik Heuristic Center*.
- The *Dyer and Freize Heuristic Center* was very fast compared to *Plesnik Heuristic Center* and all the other algorithms tried. While it does not produce a lower bound, it produced approximate solutions of a similar radius to those produced by *Plesnik Heuristic Center*, although not quite as good.
- Random search does very well for small size centers. An equivalent amount of random search found better centers than any of the other algorithms for small center sizes in the vertex unweighted cases. Random search rapidly decreased in effectiveness and did poorly on all cases other than the smallest size centers.

To present the results effectively, we measure the results for each algorithm against those of *Plesnik Heuristic Center*, and normalize it compared to the size of the lower bound / upper bound interval established by the *Plesnik Heuristic Center* algorithm. Most figures in this chapter show a result measured against this interval, normalized so that the lower bound from *Plesnik Heuristic Center* would be at 0 and the approximation found by *Plesnik Heuristic Center* would be at 1. In most cases we plot the result $X$ for a graph where the *Plesnik Heuristic Center* approximation was $U$, and the lower bound was $L$ and present it as $1 + ((X - U)/(U - L))$. Lower bounds $Y$ from Secondary Ball Heuristic Center are presented as $(Y - L)/(U - L)$.

### 5.1.1 Recomputed Ball Heuristic Center Results

Recomputed Ball heuristic center performed very poorly compared to all other algorithms except random choice. This failure seems to be due to a flaw in our perception of how the ball heuristic could help us when we designed it. The hope was that by recomputing the ball heuristic once some vertices had been labeled, we would be making better decisions than in the cases where we did not recompute the values of the ball heuristic. As it turns out this is not the case. Figure 5.3 shows typical results from The Recomputed Ball Heuristic algorithm.

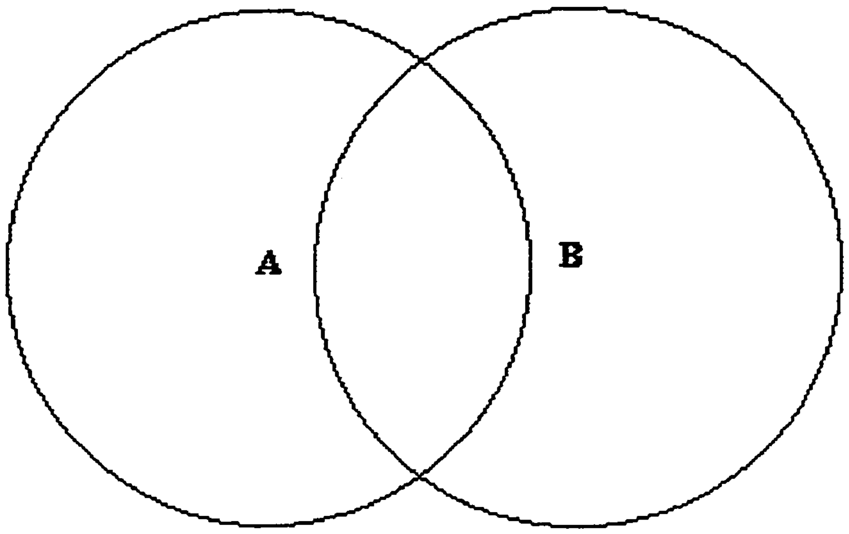

Figure 5.1: If our heuristic allows us to choose candidates that may cover the some of same vertices, we can pick vertices closer together, and hope our solution has a better radius than $2r$

All the Ball heuristic algorithms use a small radius value, $r$, in their search for a solution. When a vertex is chosen the algorithms label all vertices within $2r$ as being covered by the chosen vertex. In the usual case, this means that the algorithm can not choose a vertex within $2r$ of another vertex that has been previously chosen, but it may choose one that would also cover some of the vertices within $2r$ of a previously chosen vertex. When this happens the areas within $2r$ of the vertices will overlap as in in Figure 5.1.1. The result of this is that the center we obtain can, and usually does result in a radius less than $2r$.

Unlike the other algorithms, The Recomputed Ball Heuristic algorithm does allow vertices to be chosen within $2r$ of another previously chosen vertex. In practice however, this doesn't happen very often. When the ball heuristic values are recomputed, the heuristic values of the labeled vertices, as well as vertices near to them will drop. This means that we are much more likely to choose a vertex whose heuristic value has not dropped. This will more likely be a vertex that doesn't cover any of the same vertices as previously covered, as in Figure 5.1.1. The result of this is that the center we obtain usually has a radius that is more likely to be $2r$ or more. The algorithm chooses vertices too far apart, and the resulting center is not as good.

Some attempts was made to modify the algorithm to rectify this by only labeling vertices within $r$ (or a different constant factor of $r$) from each other. These attempts were not successful, and further work on a recomputing ball heuristic was abandoned.

### 5.1.2 Secondary Ball Heuristic Center Results

Secondary Ball Heuristic Center produced results as we predicted, which were better than *Plesnik Heuristic Center* on many graphs. These are shown in figures 5.4, 5.5, 5.6, 5.7, 5.8, and 5.9. The

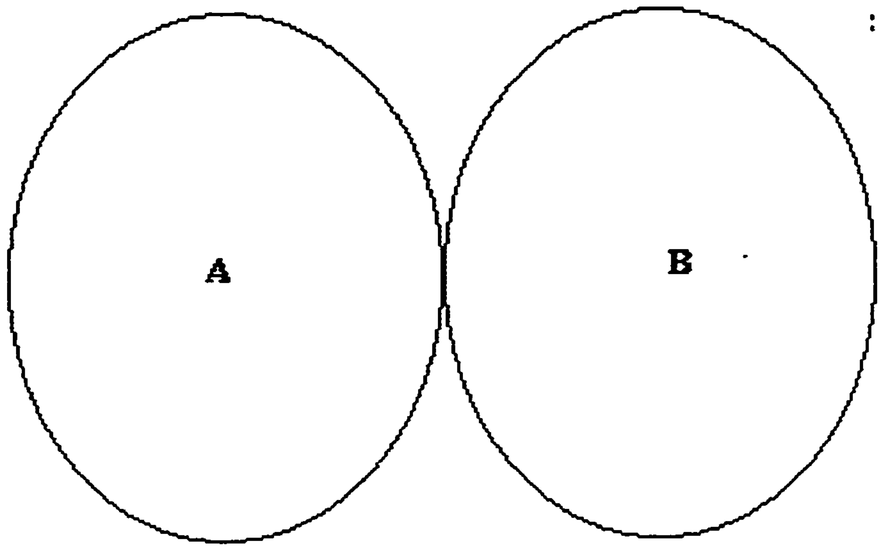

Figure 5.2: If our heuristic discourages making choices that cover the same vertices, we will be more likely to pick vertices very far away from each other

algorithm performed very well on the vertex unweighted graphs. On these graphs it normally found a much better approximation than *Plesnik Heuristic Center*, and also produced a much higher lower bound than *Plesnik Heuristic Center*, effectively narrowing the range in which we can be sure an optimal solution must lie. This was most dramatic on our vertex unweighted class of graphs with more variation in the path lengths, class number 6, as shown in Figure 5.9. It is also apparent in class 3, shown in Figure 5.6. The approximate solutions found by the algorithm in this case are almost always equal or better than those found by *Plesnik Heuristic Center*.

For graphs with vertex weights, the improvement over *Plesnik Heuristic Center* is less pronounced, with the improvement in lower bound rapidly diminishing as vertex weights are added. Even with no improvement in lower bound, such as appears to be common on the weighted graphs, the algorithm on average tends to find an approximate solution slightly better than *Plesnik Heuristic Center*, yet this is not always the case, with values ranging significantly above or below those found by *Plesnik Heuristic Center* on a per graph basis. On some graphs in each class the algorithm did substantially better, and on some it did substantially worse than *Plesnik Heuristic Center*. This is important if we consider running multiple algorithms looking for better results.

### 5.1.3 Simple Ball Heuristic Center Results

In general, Simple Ball Heuristic Center obtained similar approximate values to Secondary Ball Heuristic Center, with a few exceptions. The results are shown in figures 5.10, 5.11, 5.12, 5.13, 5.14, and 5.15. As we expected, on vertex unweighted graphs, Simple Ball Heuristic Center found exactly the same approximate solutions as did Secondary Ball Heuristic Center. On Vertex Weighted graphs it performed slightly better for the smallest (size 7) centers. Similar to Secondary

Ball heuristic Center, its results varied fairly widely, sometime the same, sometimes better, and sometimes worse than *Plesnik Heuristic Center*. These variations were not always on the same graphs for which the variations were noticed for Secondary Ball Heuristic Center on the vertex weighted graphs.

### 5.1.4 *Dyer and Freize Heuristic Center* Results

The *Dyer and Freize Heuristic Center* algorithm performed comparably to *Plesnik Heuristic Center* in most classes. As with the Simple and Secondary Ball Heuristic algorithms, it's results could vary with respect to *Plesnik Heuristic Center* on a graph by graph basis. On average it found slightly poorer solutions than *Plesnik Heuristic Center* more often than it found a better one, but was capable of finding good approximations and on average seemed to perform near *Plesnik Heuristic Center*. *Dyer and Freize Heuristic Center* is very fast compared to the other algorithms, as we would expect from the complexity results. *Dyer and Freize Heuristic Center* results are shown in figures 5.16, 5.17, 5.18, 5.19, 5.20, and 5.21.

### 5.1.5 Random Search

Random Search is included to put the performance of the heuristic approximation algorithms in an appropriate context. We want to ensure our heuristics are actually doing something for us, and our algorithm is doing better than a roughly equivalent amount of work making random guesses. The random search algorithm was implemented to make use of the computed distance graph as did all the other algorithms, in order to be able to compute the radius of a *k-set* in $O(n)$ comparisons. It was then adjusted to exit after a similar CPU usage as the slowest of the other algorithms.

Random search's approximation was quite good on the small center size of 7. It got steadily worse as the center sizes got bigger. Random search produced some of the best approximations on vertex unweighted graphs for centers of size 7. For all larger center sizes, Random search performed much worse than any of the other algorithms.

Typical performance for a vertex unweighted graph is shown in Figure 5.23, showing results for the vertex-unweighted class 6, and Figure 5.22 showing results for the vertex-weighted class 5.

## 5.2 Conclusions

Randomly choosing a center works well for small size centers. It does not work well on larger size centers when compared to a more effective approximation algorithm.

*Dyer and Freize Heuristic Center* normally produced slightly worse approximations than *Plesnik Heuristic Center*, although it occasionally found a better solution than *Plesnik Heuristic Center*.Given the simplicity and speed of the *Dyer and Freize Heuristic Center* algorithm, it should be a good choice in general when a lower bound is not needed, and an approximate solution is desired with low computational cost. *Dyer and Freize Heuristic Center* was normally 10 times faster than *Plesnik Heuristic Center* on the graphs in our tests. Given the complexity of the algorithms this difference will increase on larger graphs. In our experiment we used an initial run of *Plesnik Heuristic Center* to establish a lower bound on the optimal center radius, which we then used as the "inexpensive" radius at which to compute the Ball Heuristic values for the Ball Heuristic algorithms. *Dyer and Freize Heuristic Center* could potentially be used instead - using a value somewhere around half, or a little less than half of the radius returned by *Dyer and Freize Heuristic Center* as the value at which to compute the ball heuristic. This value would likely be near the lower bound value of the *Plesnik Heuristic Center* algorithm, and while such a value would not be a provable lower bound on the optimal solution, it would work well in practice in most cases.

On vertex unweighted graphs, Secondary Ball Heuristic Center produces a much better solution than *Plesnik Heuristic Center.* The higher lower bound and a better approximate solution make it a better choice for unweighted graphs, and in all cases it retains the provably 2-approximate property of *Plesnik Heuristic Center.* Secondary Ball Heuristic Center appears to be a good choice in general to get the best approximation possible from a single algorithm.

With all the algorithms, the per-graph difference from *Plesnik Heuristic Center* could vary substantially. On most classes, an algorithm that did better on average or even most of the time did not mean that it did better all of the time. Similarly, even algorithms that in general produced worse approximations than *Plesnik Heuristic Center* occasionally found better solutions. Considering the similarities in these algorithms and the cost of the precomputation (the Distance Graph) that is needed for all of them, a strong case can be made for running multiple algorithms and choosing the best approximation out of all of them as an effective technique for approximating *k-center* in general.

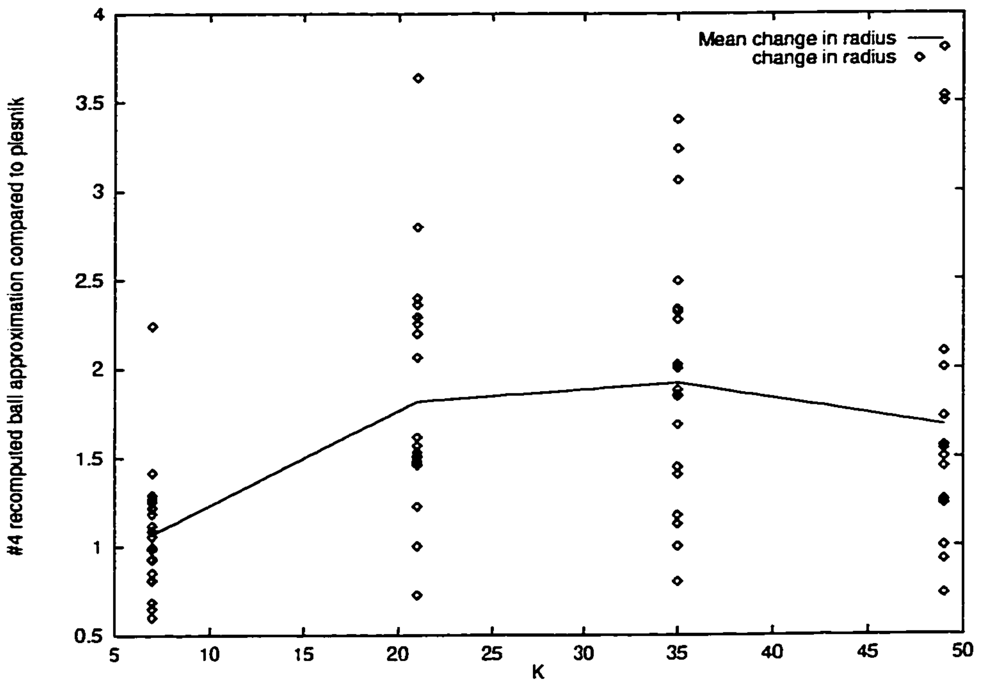

Figure 5.3: Recomputed Ball Heuristic approximation on 20 graphs from class 4, normalized against *Plesnik Heuristic Center* approximation at 1, and lower bound at 0

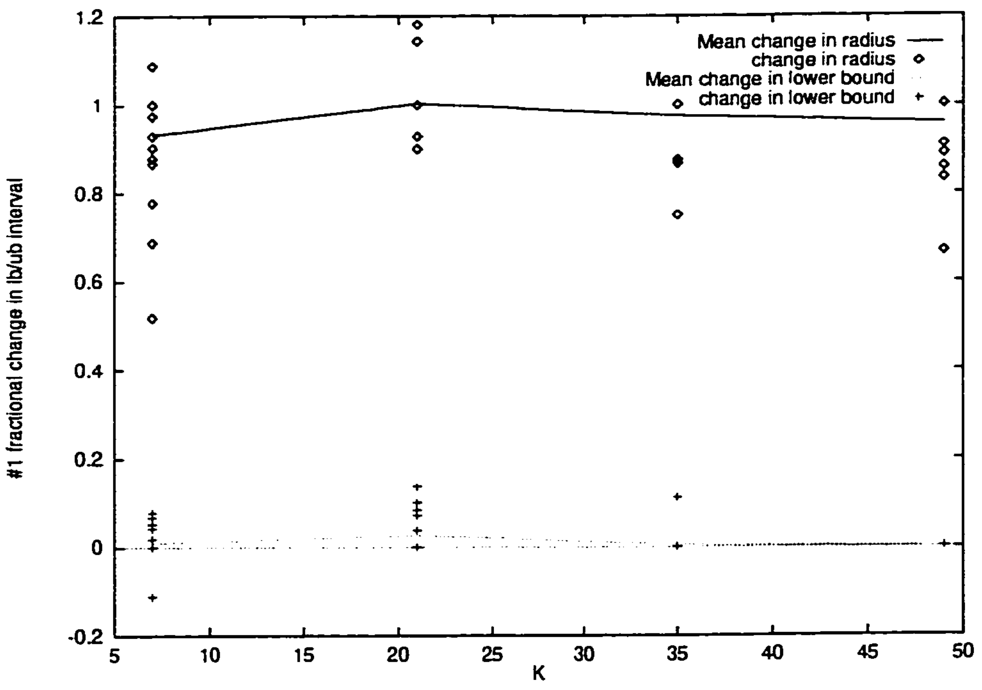

Figure 5.4: Secondary Ball Heuristic approximation and lower bound on 20 graphs from class 1, normalized against *Plesnik Heuristic Center* approximation at 1, and lower bound at 0

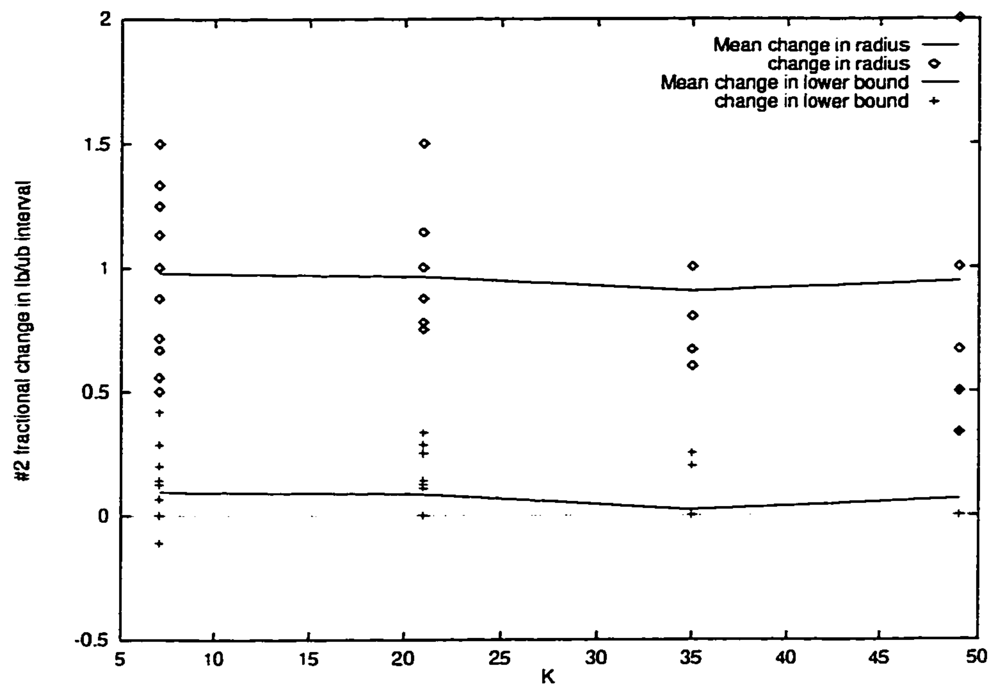

Figure 5.5: Secondary Ball Heuristic approximation and lower bound on 20 graphs from class 2, normalized against *Plesnik Heuristic Center* approximation at 1, and lower bound at 0

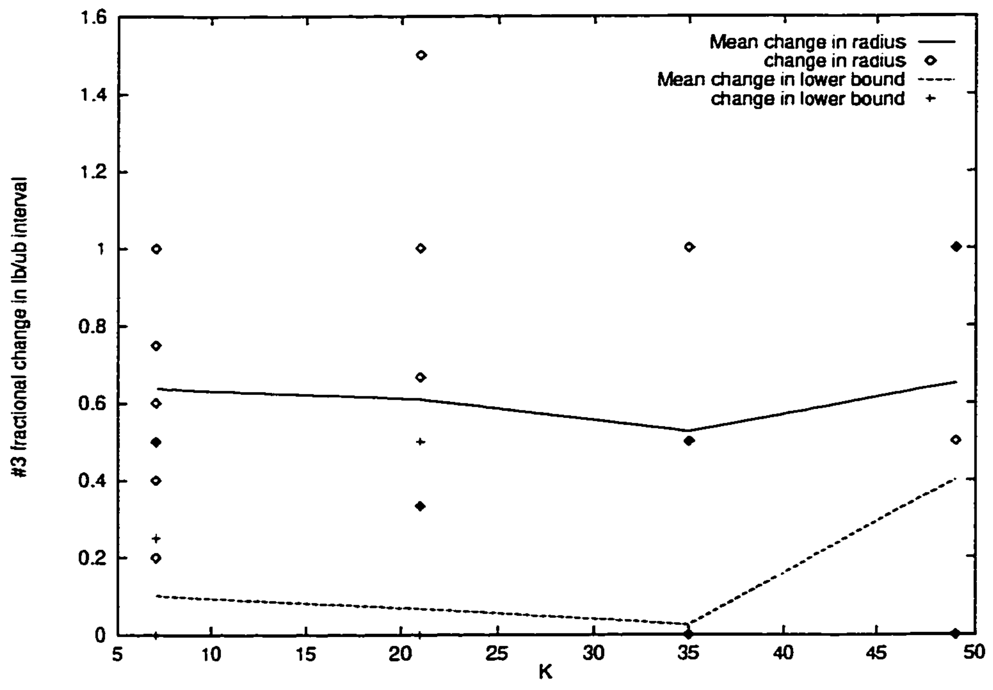

Figure 5.6: Secondary Ball Heuristic approximation and lower bound on 20 graphs from class 3, normalized against *Plesnik Heuristic Center* approximation at 1, and lower bound at 0

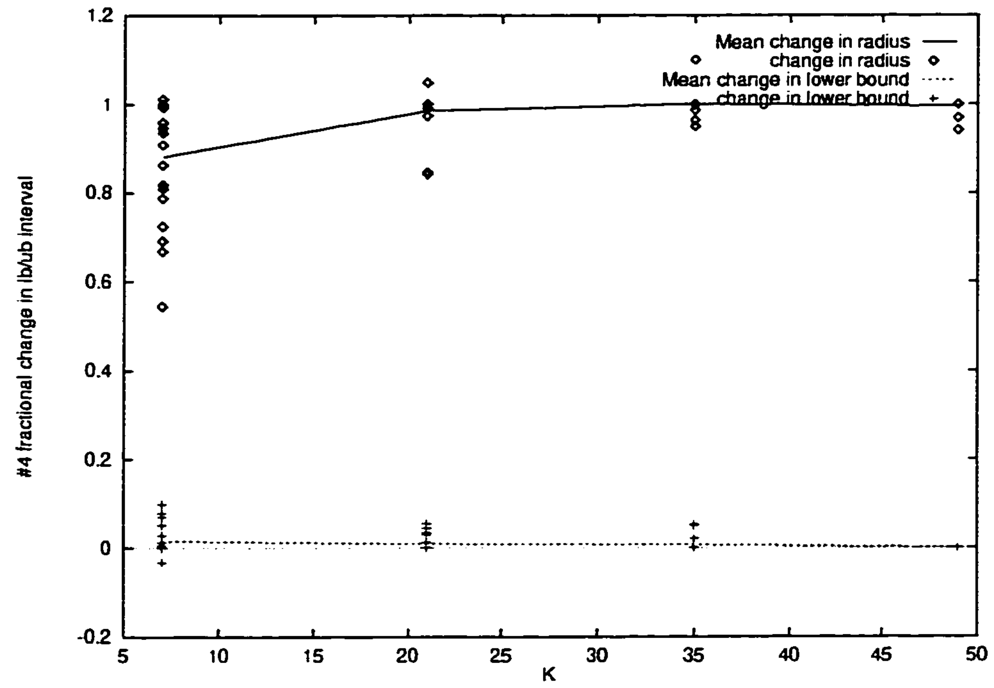

Figure 5.7: Secondary Ball Heuristic approximation and lower bound on 20 graphs from class 4, normalized against *Plesnik Heuristic Center* approximation at 1, and lower bound at 0

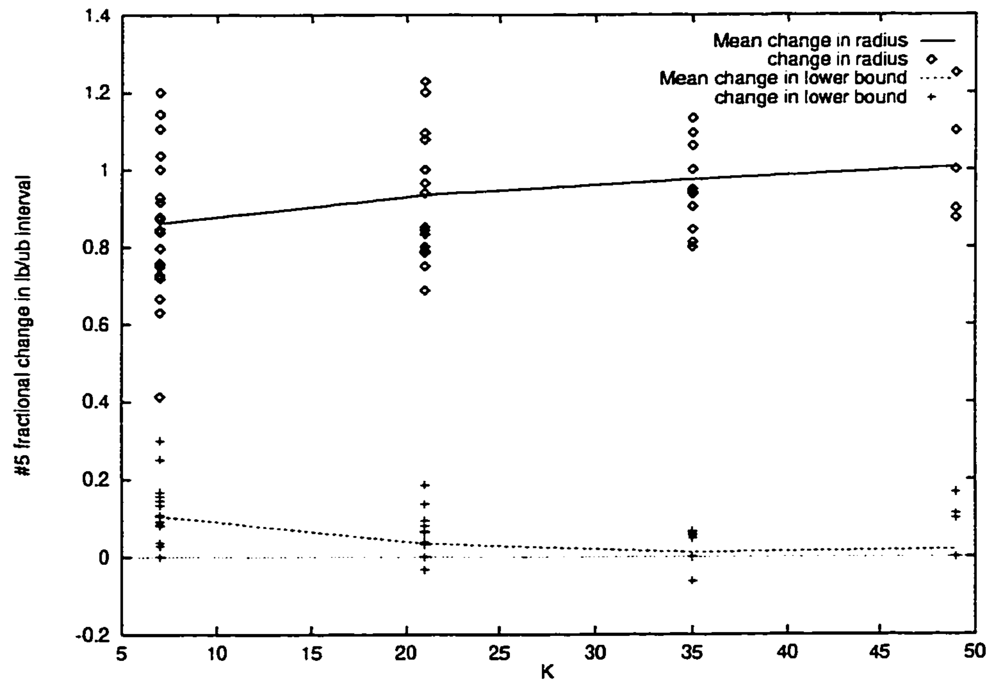

Figure 5.8: Secondary Ball Heuristic approximation and lower bound on 20 graphs from class 5, normalized against *Plesnik Heuristic Center* approximation at 1, and lower bound at 0

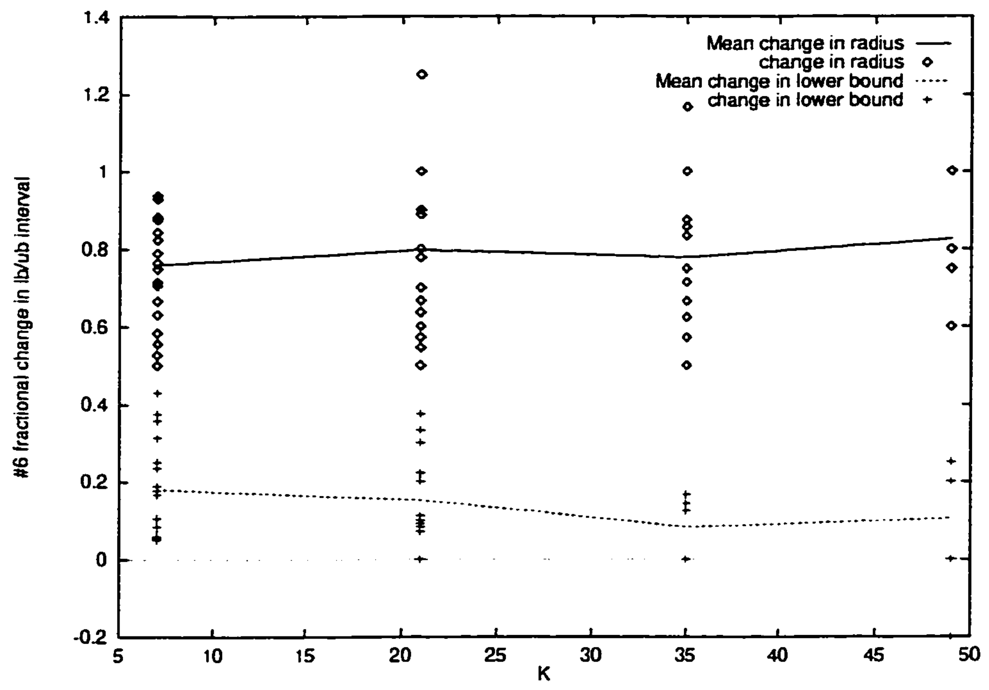

Figure 5.9: Secondary Ball Heuristic approximation and lower bound on 20 graphs from class 6, normalized against *Plesnik Heuristic Center* approximation at 1, and lower bound at 0

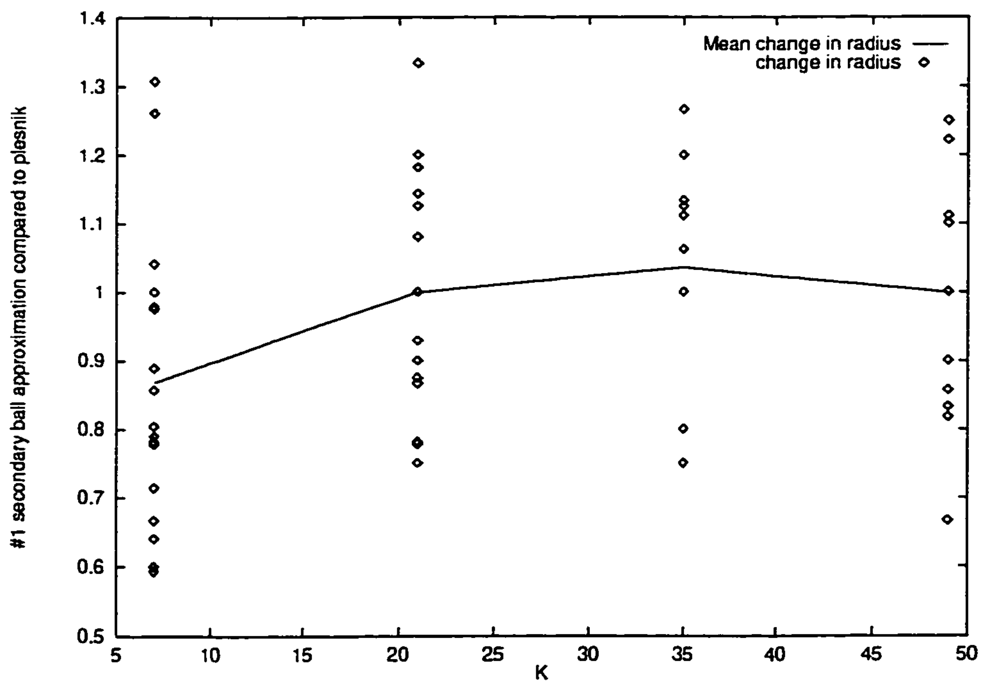

Figure 5.10: Simple Ball Heuristic approximation on 20 graphs from class 1, normalized against *Plesnik Heuristic Center* approximation on a per graph basis. *Plesnik Heuristic Center* approx would be at 1, lower bound at 0.

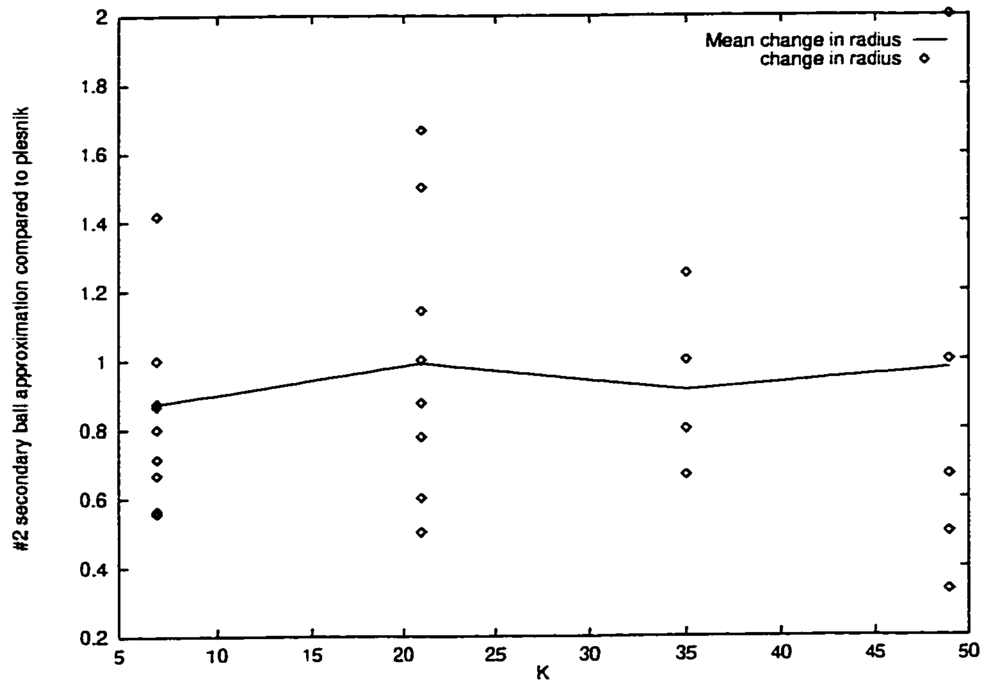

Figure 5.11: Simple Ball Heuristic approximation on 20 graphs from class 2, normalized against *Plesnik Heuristic Center* approximation at 1, and lower bound at 0

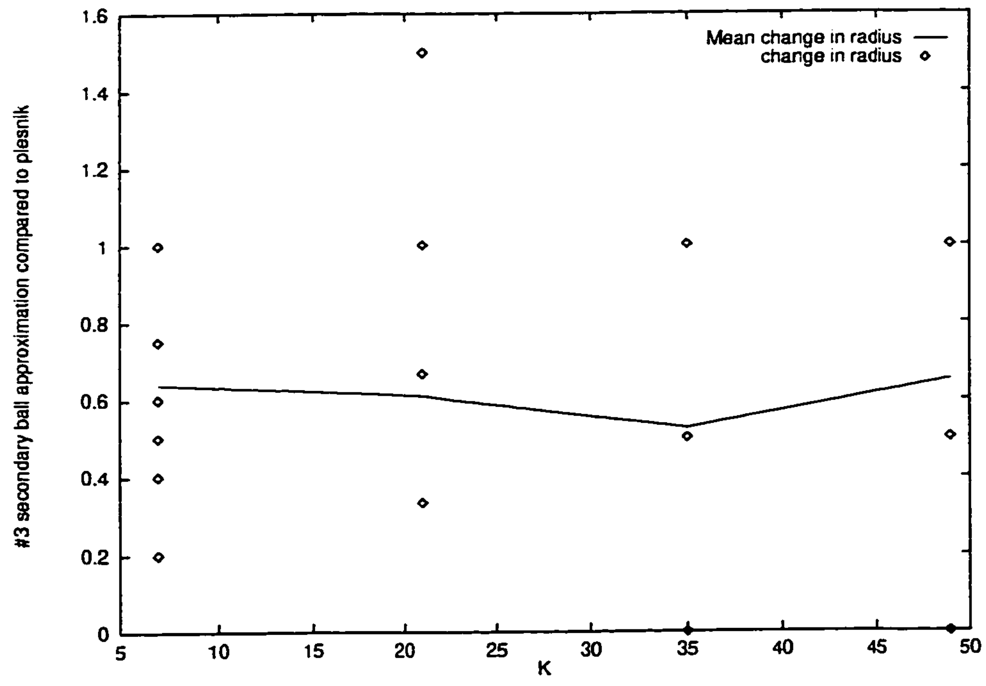

Figure 5.12: Simple Ball Heuristic approximation on 20 graphs from class 3, normalized against *Plesnik Heuristic Center* approximation at 1, and lower bound at 0

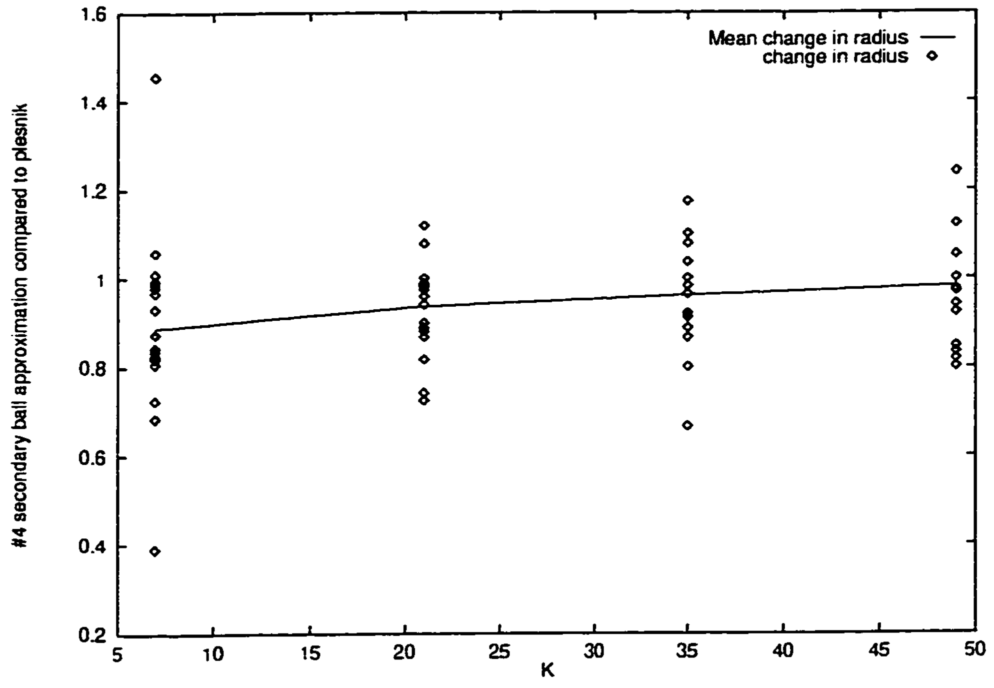

Figure 5.13: Simple Ball Heuristic approximation on 20 graphs from class 4, normalized against *Plesnik Heuristic Center* approximation at 1, and lower bound at 0

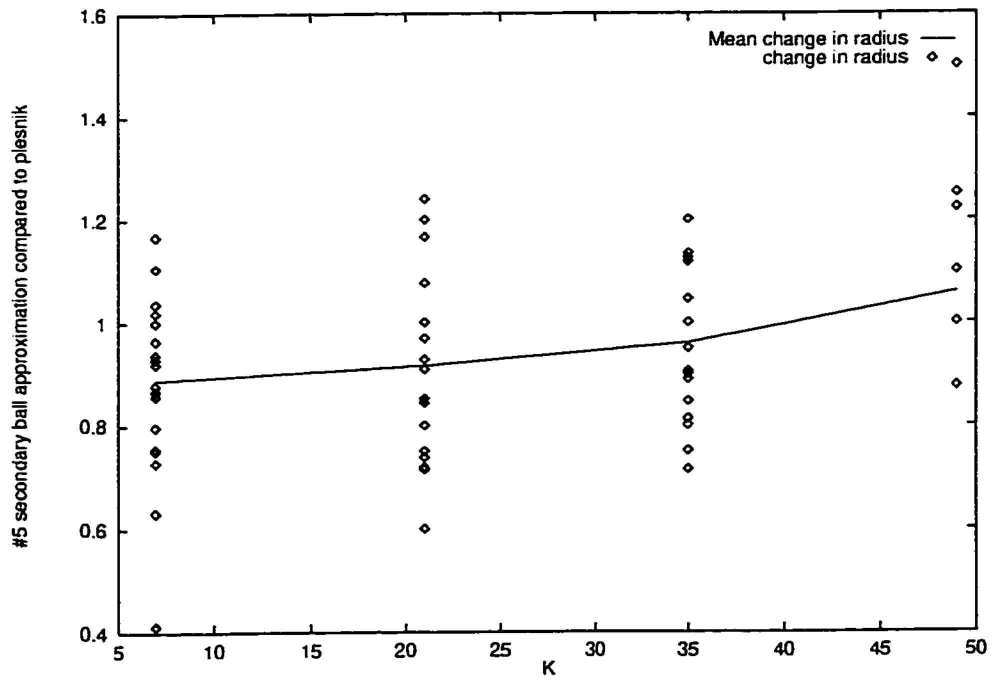

Figure 5.14: Simple Ball Heuristic approximation on 20 graphs from class 5, normalized against *Plesnik Heuristic Center* approximation at 1, and lower bound at 0

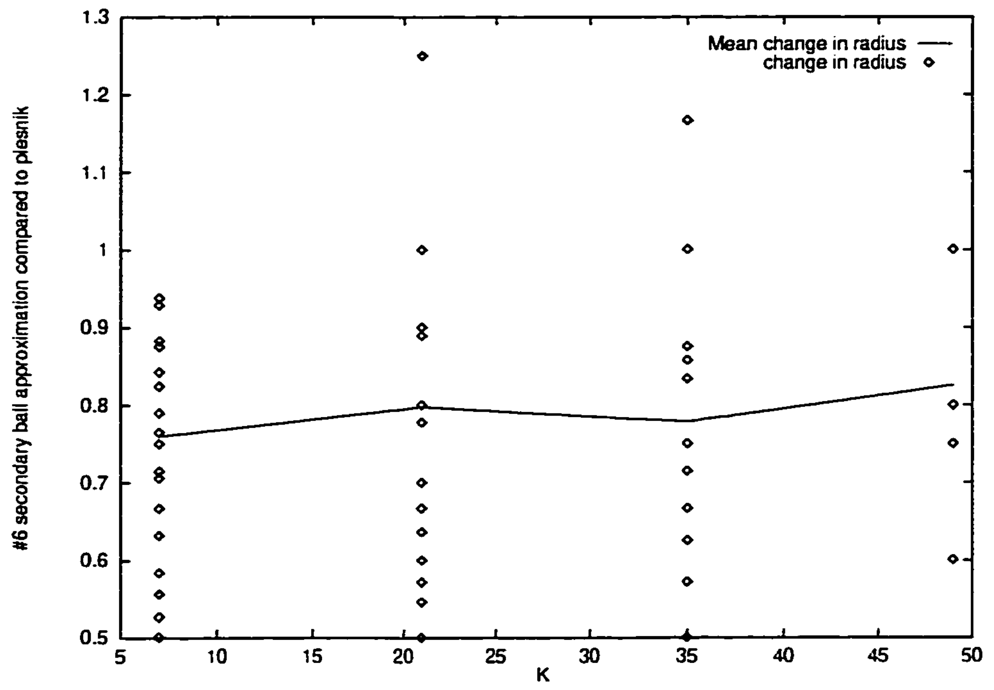

Figure 5.15: Simple Ball Heuristic approximation on 20 graphs from class 6, normalized against *Plesnik Heuristic Center* approximation at 1, and lower bound at 0

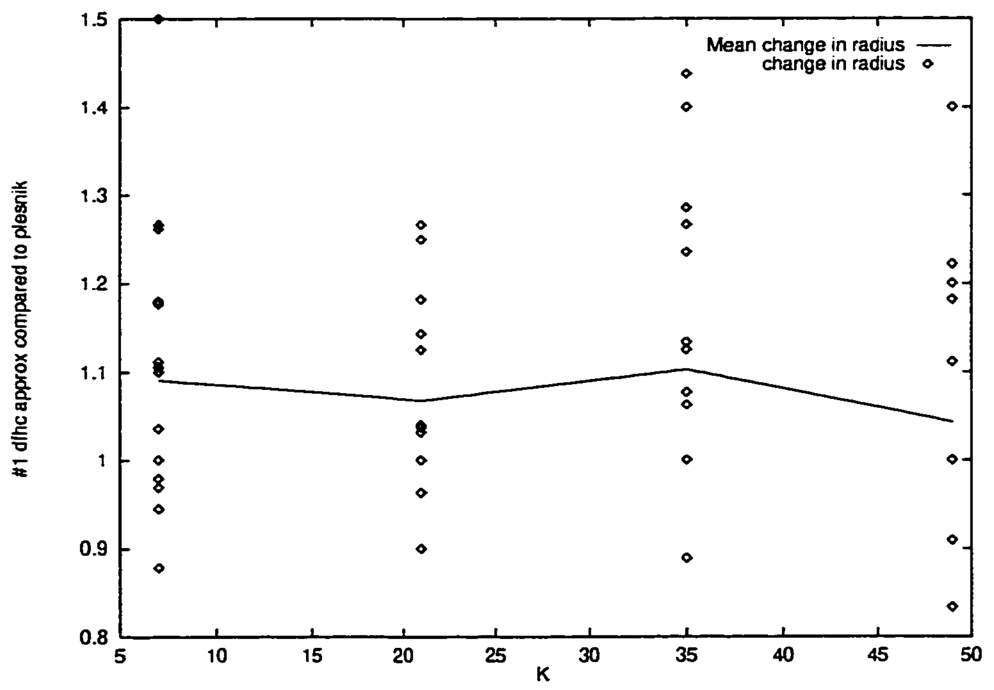

Figure 5.16: *Dyer and Freize Heuristic Center* approximation on 20 graphs from class 1, normalized against *Plesnik Heuristic Center* approximation at 1, and lower bound at 0

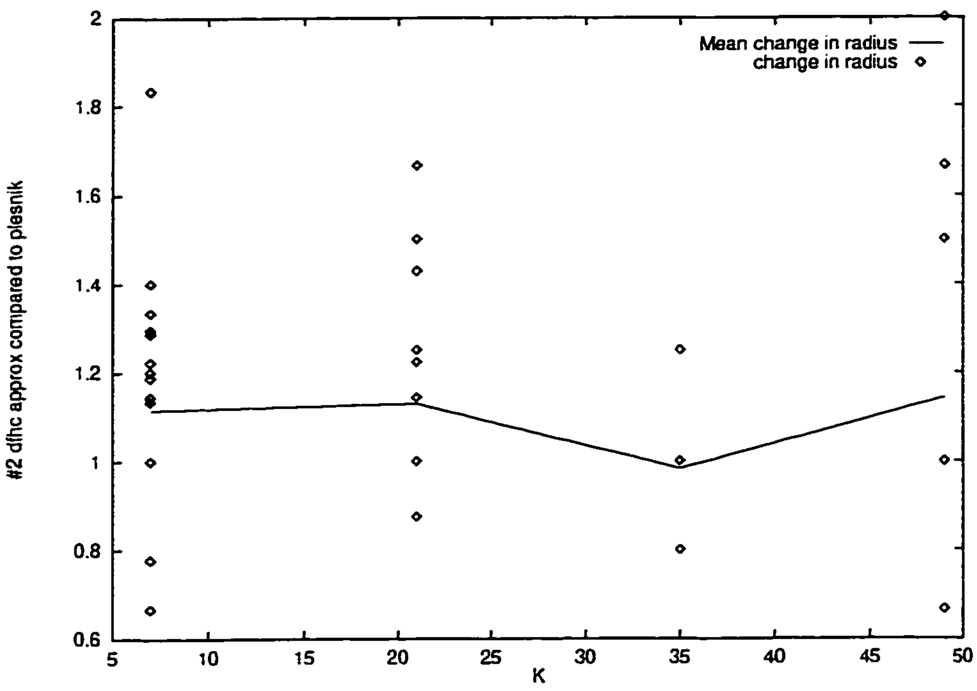

Figure 5.17: *Dyer and Freize Heuristic Center* approximation on 20 graphs from class 2, normalized against *Plesnik Heuristic Center* approximation at 1, and lower bound at 0

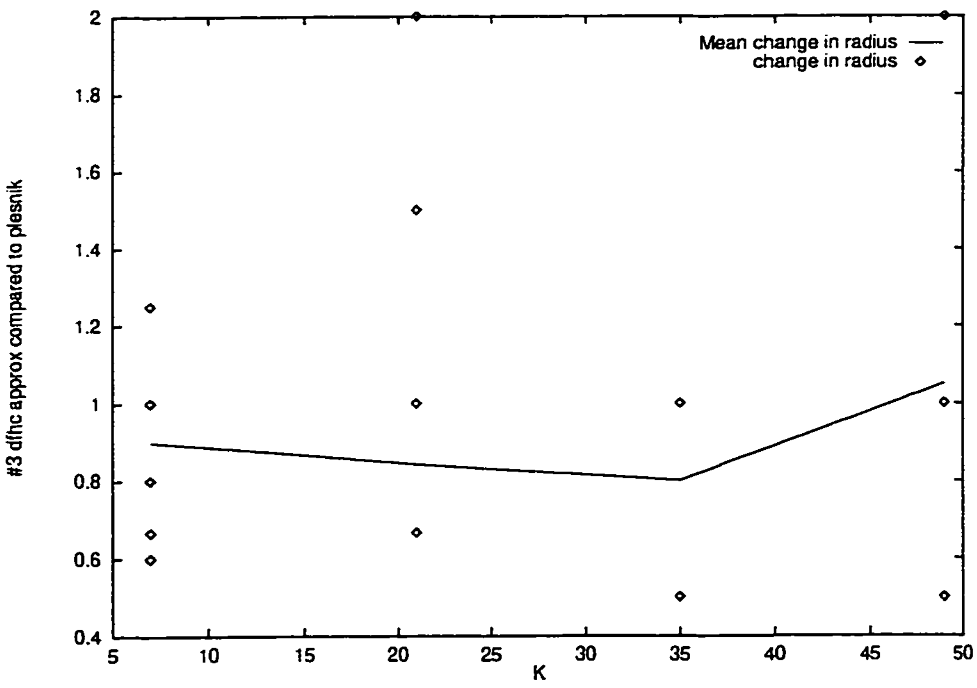

Figure 5.18: *Dyer and Freize Heuristic Center* approximation on 20 graphs from class 3, normalized against *Plesnik Heuristic Center* approximation at 1, and lower bound at 0

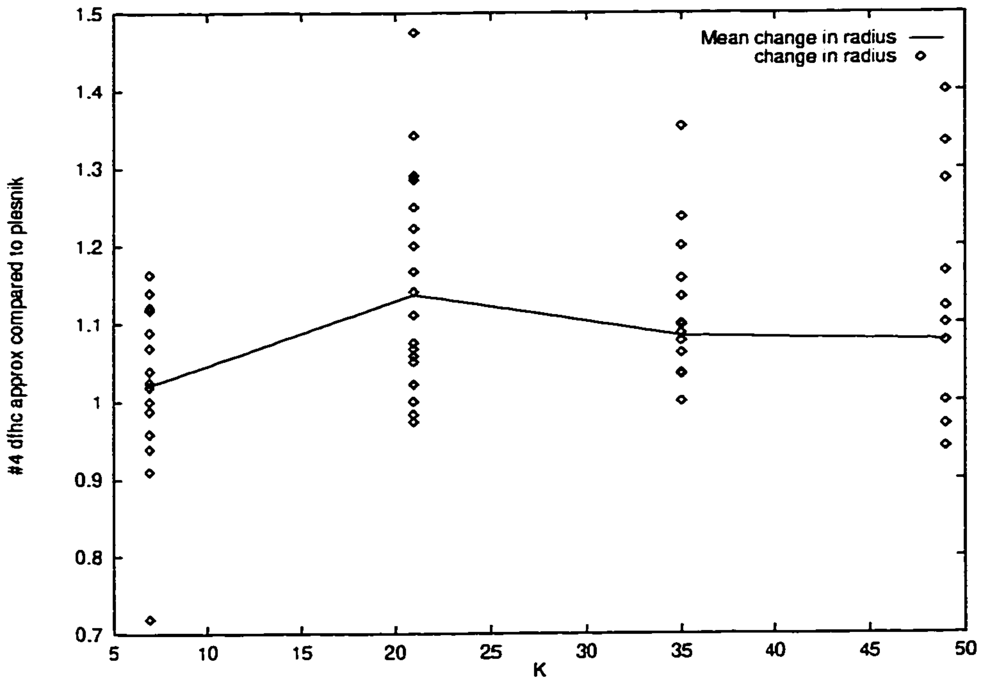

Figure 5.19: *Dyer and Freize Heuristic Center* approximation on 20 graphs from class 4, normalized against *Plesnik Heuristic Center* approximation at 1, and lower bound at 0

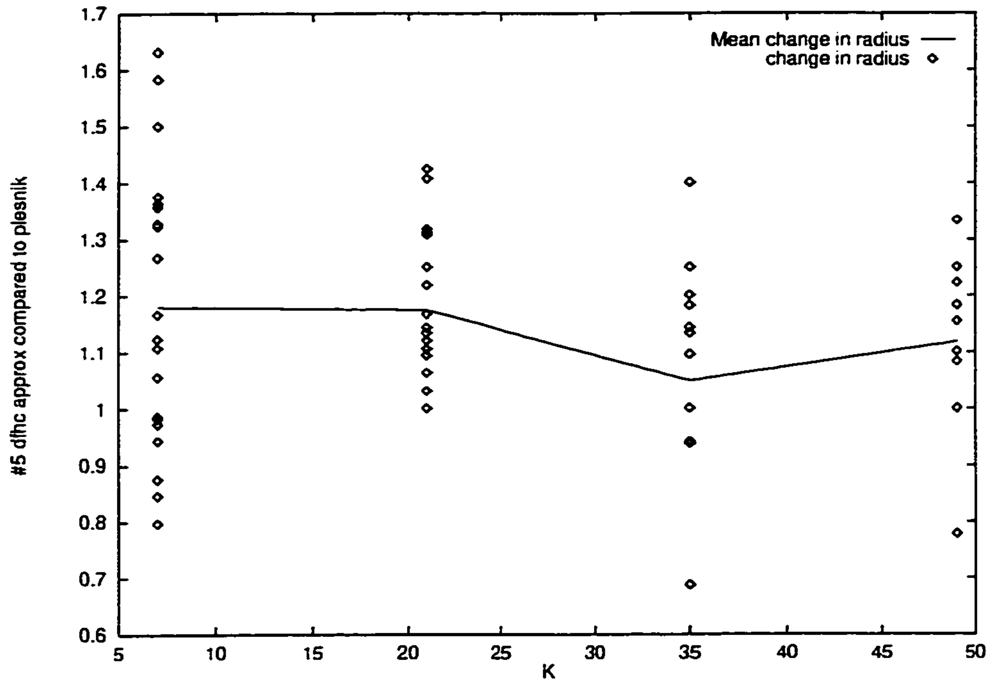

Figure 5.20: *Dyer and Freize Heuristic Center* approximation on 20 graphs from class 5, normalized against *Plesnik Heuristic Center* approximation at 1, and lower bound at 0

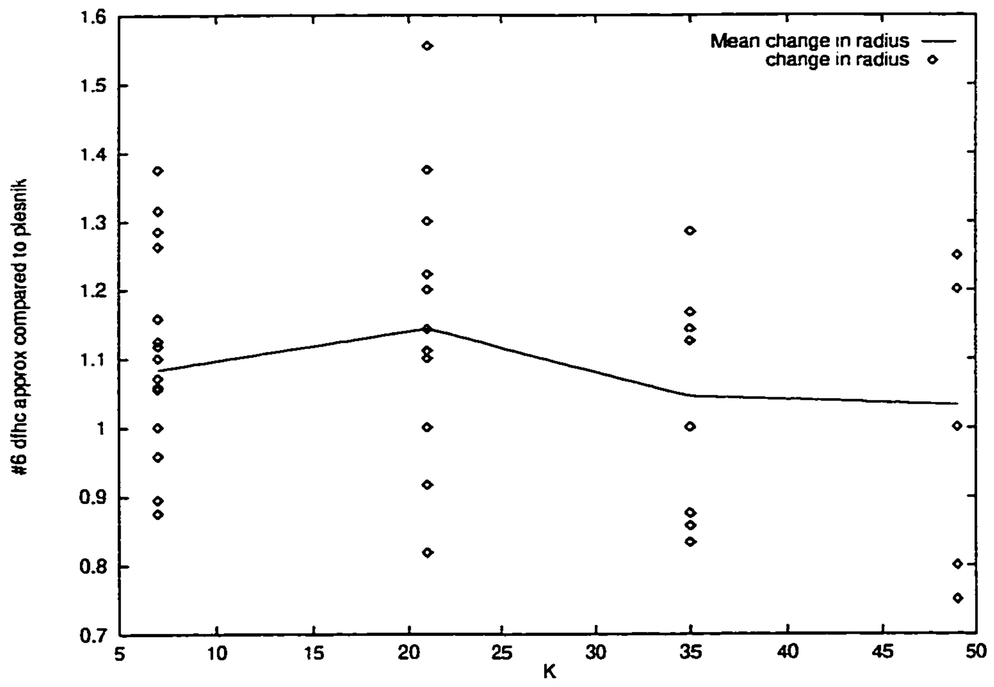

Figure 5.21: *Dyer and Freize Heuristic Center* approximation on 20 graphs from class 6, normalized against *Plesnik Heuristic Center* approximation at 1, and lower bound at 0

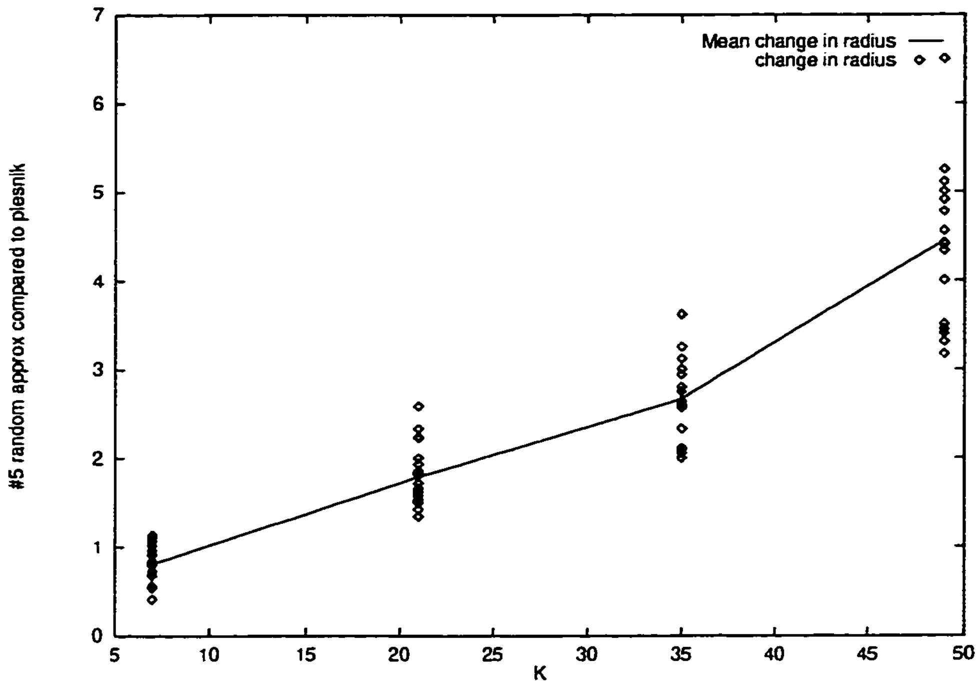

Figure 5.22: Random Search approximation on 20 graphs from class 5, normalized against *Plesnik Heuristic Center* approximation at 1, and lower bound at 0

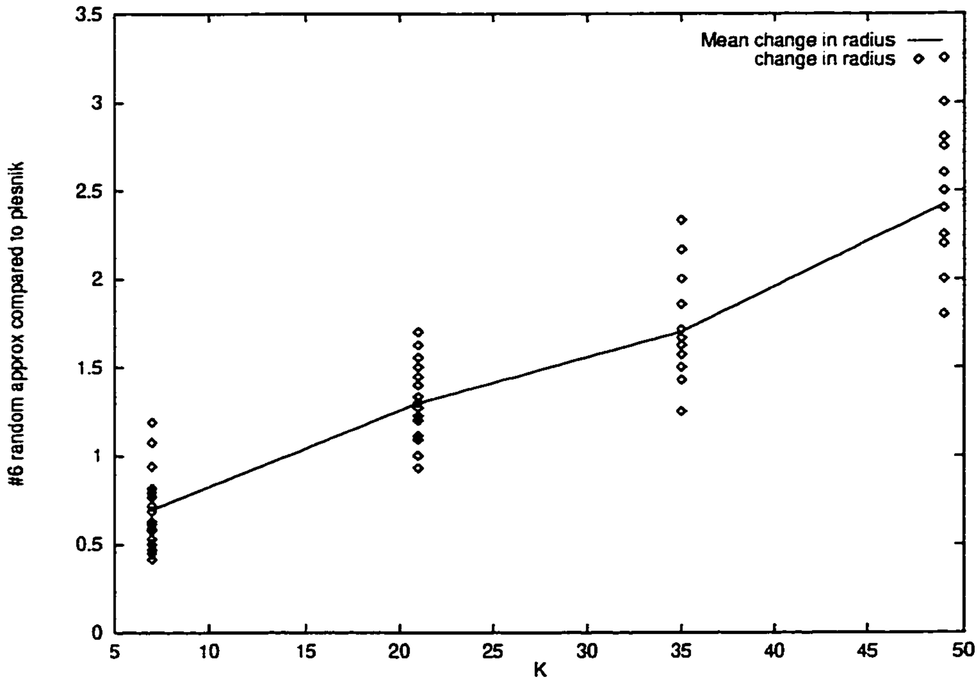

Figure 5.23: Random Search approximation on 20 graphs from class 6, normalized against *Plesnik Heuristic Center* approximation at 1, and lower bound at 0

# Chapter 6

# Conclusions

Exact solutions to the *k-center* problem are predictably difficult in the general case. We have found however, that by using a pruning search technique, a significant amount of work can be saved. A pruning exact search is possible on smaller sized graphs for small size centers, and makes an effective approximation technique if we do not allow it to run to completion. Our experiments showed that over multiple randomly generated instances, the distribution of search times is such that many solutions are found relatively quickly, with only some instances taking a very large amount of search. This seems to indicate that a pruning search, or multiple independent searches will be beneficial if we must have an exact solution, or wish to use a limited search method as a means of obtaining an approximate solution.

We tested and examined the performance of the *Plesnik Heuristic Center* algorithm on varying classes of random inputs, compared it's values both to exact solutions, and to other approximation techniques.

When finding approximate solutions to the *k-center* problem, we found that the existing approximation algorithms could be improved considerably at some computational cost. On vertex unweighted graphs, the Secondary Ball Heuristic Center algorithm offered a significant advantage over the traditional 2-approximate algorithms, with an increase on average in the lower bound on the optimal solution, and a better approximate solution found. For weighted graphs we found that the clear advantage of this type of algorithm went away as the distribution of the vertex weights in the graph became more varied, with fewer vertices sharing a common weight.

In addition to finding improved approximation algorithms, our experimental results show that different approximate solutions can be found by different algorithms on the same graphs. This indicates that running multiple differing approximation algorithms and choosing the best result we can obtain will be beneficial to us if we can justify the increased computational time with a better approximate solution.

## 6.1 Further Work

The Ball heuristics we used were based on vertex weight totals. Are there other heuristics that can be easily computed that give us a useful measure of the nearby vertices to a candidate? Can they be used in an algorithm similar to those we have described?

In chapter 4, we established that to guarantee a 2-approximate solution, a greedy selection strategy similar to procedure RANGE from *Plesnik Heuristic Center* does not need need to choose a vertex whose weight is maximal over the entire graph, but rather is maximal over all vertices within distance $2r$ from itself. We also establish a bound on the optimality of a solution found if we choose a vertex that is not a maximum weight vertex. Can these facts be capitalized on more directly to obtain better solutions?

Is there another heuristic that could be applied to vertex selection in a *k-center* algorithm that would give us an estimate of the cost to not include the vertex in the center?

On what other classes of random graph may it be easier/harder to obtain solutions using *Depth First Center* or the *Plesnik Heuristic Center* algorithm? Can we generate different classes of graph that exaggerate or shrink the performance characteristics we have seen?

The *Dyer and Freize Heuristic Center* algorithm chooses vertices by max weight initially, and then by the furthest vertex from the center in construction. Is there a way to modify the *Dyer and Freize Heuristic Center* algorithm to choose different vertices and locate a better center on average? The *Dyer and Freize Heuristic Center* algorithm is also very fast. It is likely possible, just by changing the algorithm's choice of the first vertex to affect the end result. It may be worthwhile to attempt an algorithm that consists of multiple *Dyer and Freize Heuristic Center* searches, while permuting or modifying the algorithm's starting vertex, and possibly it's strategy for adding vertices to the center. Choosing vertices of large Ball Heuristic value as initial start points may be a useful modification.

We found a large improvement in the vertex unweighted case of the problem using our Secondary Ball Heuristic Center algorithm. For the purposes of the *k-center* problem a vertex weighted graph may be transformed to an equivalent vertex unweighted directed graph where the edges are weighted as the weighted distance in the original graph. Can this fact be used to apply Secondary Ball Heuristic Center more successfully to vertex weighted graphs? Will such an algorithm be 2-approximate?

# Bibliography

[1] Ken McAloon Carla P. Gomes, Bart Selman and Carol Tretkoff. Randomization in backtrack search: Exploiting heavy-tailed profiles for solving hard scheduling problems. *Proceedings AIPS98 Pittsburg PA*, 1998.

[2] M.E. Dyer and A.M. Frieze. A simple heuristic for the p-centre problem. *Operations Research Letters*, 3:285 - 288, 1985.

[3] A. H. G. Rinnooy Kan E. L. Lawler, J. K. Lenstra and D. B. Schmoys. *The Travelling Salesman Problem, A Guided Tour of Combinatorial Optimization.* John Wiley and Sons, New York, 1985.

[4] M.R. Garey and D.S. Johnson. *Computers and Intractablilty, A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

[5] Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, Boston, Massachusetts, 1996.

[6] D.S. Hochbaum. Easy solutions for the $k$-center problem or the dominating set problem on random graphs. *Annals of Discrete Mathematics*, 25:189 - 209, 1985.

[7] D.S. Hochbaum and D.B. Shmoys. A best possible heuristic for the $k$-center problem. *Mathematics of Operations Research*, 10(2):180 - 184, 1985.

[8] W. Hsu and G. L. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1:209 - 215, 1979.

[9] J.S.Martinich. A vertex-closing approach to the $p$-center problem. *Naval Research Logistics*, 35:185 – 201, 1988.

[10] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. i: The $p$-centers. *SIAM J. of Applied Mathmatics*, 37(3):513 - 537, 1979.

[11] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. ii: The $p$-medians. *SIAM J. of Applied Mathmatics*, 37(3):539 - 560, 1979.

[12] Catherine C McGeoch. Toward an experimental method for algorithm simulation. *INFORMS Journal on Computing*, 8:1 - 28, 1996.

[13] S.E. Nikoletseas and P.G. Spirakis. Near optimal dominating sets in dense random graphs in polynomial expected time. *Lecture Notes in Computing Science*, 790:1 - 10, 1993.

[14] Edgar M. Palmer. *Graphical Evolution*. John Wiley and Sons, New York, 1985.

[15] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization; Algorithms and Complexity*. Prentice Hall Inc., Englewood Cliffs, New Jersey 07632, 1982.

[16] J. Plesník. A heuristic for the $p$-center problem in graphs. *Discrete Applied Mathematics*, 17:263 - 268, 1987.

# IMAGE EVALUATION TEST TARGET (QA-3)