2011

A Domain Aware Genetic Algorithm for the p-Median Problem

Dennis Vickers
Nova Southeastern University, davickers@gmail.com

A Domain Aware Genetic Algorithm for the *p*-Median Problem

by

Dennis Vickers
e-mail: vdennis@nova.edu
voice: (805) 377-0246

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in
Computer Information Systems

Graduate School of Computer and Information Sciences
Nova Southeastern University

September 2011

We hereby certify that this dissertation submitted by Dennis Vickers, conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

Sumitra Mukherjee, Ph.D. Chairperson of Dissertation Committee	Date
Michael Laszlo, Ph.D. Dissertation Committee Member	Date
Greg Simco, Ph.D. Dissertation Committee Member	Date

Approved:

Amon B Seagull, Ph.D. Dean	Date

Graduate School of Computer and Information Sciences
Nova Southeastern University
2011

An Abstract of a Dissertation Submitted to
Nova Southeastern University
in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

A Domain Aware Genetic Algorithm for the *p*-Median Problem

By
Dennis Vickers

September 2011

The *p*-median problem is an NP-complete combinatorial optimization problem often used in the fields of facility location and clustering. Given a graph with n nodes and an integer $p < n$, the *p*-median problem seeks a set of p medians such that the sum of the distances of the n nodes from their nearest median is minimized. This dissertation develops a genetic algorithm that generates solutions to the *p*-median problem that improves on previously published genetic algorithms by implementing operators that exploit domain specific information. More specifically, this GA explores the following:

(1) The advantages of using "good" solutions generated using extant heuristics in the initial generation of chromosomes.
(2) The effectiveness of a crossover operation that exchanges centers in the same neighborhood rather than exchanging arbitrarily chosen subsets of centers.
(3) The efficacy of using a biased mutation operator that favors replacing existing medians from less fit chromosomes with non-median nodes from the same neighborhood as the median being replaced.

Using published problem sets with known solutions, this dissertation examines solutions identified by the new genetic algorithm in order to determine the accuracy, efficiency and performance characteristics of the new algorithm. In addition, it tests the contribution of each of the algorithm's operators by systematically controlling for all the other factors.

The results of the analysis showed that integrating operators that exploited domain specific information did have an overall positive impact on the genetic algorithm. In addition, the results showed that using a structured initial population had little impact on the algorithm's ability to find an optimal solution but it did create a better initial solution and allowed the algorithm to produce a relatively good solution early in the search. Also, the analysis showed that a directed approach to crossover operations was effective and produced superior solutions. Finally, the analysis showed that a directed approach toward mutation did not have a large impact on the overall functionality of the algorithm and may be inferior to an arbitrary approach to mutation.

Acknowledgements

I would like to express my appreciation of Sumitra Mukherjee for persevering with me as my advisor all the way through the time it took me to complete this research and write this dissertation. The inspiration for doing the research came from the courses he taught and the discussions we had during the time I spent as a graduate student at Nova Southeastern University. He was unfailingly generous with his time and encouragement. His patience with my many delays and willingness to listen to my problems and help me focus on the solutions was invaluable. I am grateful to him and the administration at Graduate School of Computer and Information Sciences that was willing to put in the extra effort that made it possible for me to complete my degree from a geographical distance of 3,000 miles.

The members of my dissertation committee, Michael Laszlo and Greg Simco, have generously given their time and expertise to better my work. I thank them for their contribution and their generous support.

I would also like to single out my son, Scott Vickers, for his invaluable help and insight in programming the algorithm that formed a basis for this dissertation. His expertise in object oriented programming and C++ made writing the software significantly easier than it otherwise would have been.

I would especially like to express my deepest appreciation to my wife, Wendy Vickers. Not only for her love and support in what must have seemed to her for such a long time as an endless project, but also for her tireless and expert editing on a subject that in all honestly held very little interest for her but which fascinated me.

Finally, I would like to thank my family, friends and colleagues that continued to encourage me and support me throughout this whole journey.

Table of Contents

List of Tables

Tables

List of Figures

Figures

Chapter 1

Introduction

Problem Statement and Goals

This dissertation presents a new genetic algorithm for the *p*-median problem. The *p*-median problem is a graph theory problem used extensively in the field of discrete location theory for facility location analysis. In the *p*-median problem, defined on a complete directed graph with *n* nodes, *p* facilities have to be located on a graph such that the sum of Euclidian distances between the nodes of the graph and the facilities is minimized (Hakimi, 1964, 1965). This is often referred to as a "minisum" problem. A distinguishing characteristic of the *p*-median problem is that the facilities (medians) must be selected from existing points in the problem set. The *p*-median problem has been shown to be an NP-hard problem (Megiddo & Supowits, 1984) and becomes computationally intractable as the problem size increases. There has been a significant amount of research on metaheuristic approaches to the *p*-median problem (Mladenovi, Brimberg, Hansen, & Moreno-Pérez., 2007; Reese, 2005) with widely varying degrees of success (Alba & Dominguez, 2006). One approach in particular, genetic algorithms, has been only lightly studied as applied to the *p*-median problem, but shows some promise (Alp, Erkut, & Drezner, 2003; Bozkaya, Zhang, & Erkut, 2002; Chiou & Lan, 2001; Correa, Steiner, Freitas, & Carnieri, 2001; Dibble & Densham, 1993; Estivill-Castro &

Torres-Velázquez, 1999; Hosage & Goodchild, 1986). This dissertation examines the impact of integrating domain knowledge into a genetic algorithm as applied to the *p*-median problem.

The *P*-median Problem

The *p*-median problem requires the selection of *p* objects to serve as centers (or medians) for their partition. The goal is to choose medians and assign all objects to their nearest median with the objective of minimizing the sums of the distances between the centers and objects in their partition. An important aspect of the *p*-median problem is that the median of each partition is an actual object. ReVelle and Swain (1970) provided an integer programming formulation for the discrete *p*-median problem, given in Figure 1.

Like many problems of combinatorial data analysis, *p*-median has been shown to be NP-hard (Megiddo & Supowits, 1984) for an arbitrary *p*. The number of feasible solutions for the *p*-median problem is $N!/(p!(N-p)!)$. For example, if $N = 100$ and $p = 2$, there are only 4950 feasible solutions, which could easily be enumerated. However, if $N = 100$ and $p = 10$, there are more than 17 trillion solutions. This highlights one of the characteristics of the *p*-median problem, which is that as the size of the problem instance increases, it rapidly becomes too large for total enumeration.

Heuristic Approaches To the *P*-median Problem

Given the size characteristic of realistic *p*-median problems, researchers have developed heuristics that are capable of yielding good quality solutions without proof of

Minimize $\sum_{i=1}^{n}\sum_{j=1}^{n} w_i d_{ij} x_{ij}$. (1)

Subject to $\sum_{j=1}^{n} x_{ij} = 1, \quad \forall i,$ (2)

$\sum_{j=1}^{n} y_j = p$, (3)

$x_{ij} \leq y_j, \ \forall i,j,$ (4)

$x_{ij} = 0 \ or \ 1, \ \forall i,j,$

$y_j = 0 \ or \ 1, \ \forall j,$

Where

$n =$ total number of demand points,
$p =$ number of medians,
$w_i =$ demand at point i,
$d_{ij} =$ distance between points i and j,
$x_{ij} = \begin{cases} 1 \text{ if } i \text{ is assigned to median } j, \\ 0 \text{ otherwise,} \end{cases}$
$y_j = \begin{cases} 1 \text{ if the vertex } j \text{ is a median,} \\ 0 \text{ otherwise.} \end{cases}$

Condition (2) prevents a demand point *i* from being free, i.e. not having an assigned median. Condition (3) establishes the number of medians. The last condition (4) ensures the coherence of the solutions, that is, a demand point *i* cannot be assigned to the median *j* ($y_j = 1$), which is not established as median ($y_j = 0$).

Figure 1. P-median Problem Formulation

their optimality, in a practical time (Mladenovi, et al., 2007). Two heuristics that are promising are the Tabu Search heuristic (Rolland, Schilling, & Current, 1996) and the Variable Neighborhood Search (VNS) heuristic (Hansen & Mladenovic, 1997). These heuristics have several common characteristics that allow them to exploit promising local search areas without sacrificing exploration of the global search space. They use a structured search space, made up of multiple "neighborhoods". Though they have differing definitions of a neighborhood, they each use a local search within a neighborhood to concentrate on promising solutions. They both have methods for moving the search outside of a neighborhood to minimize the risk of being trapped at a local

optimum. This dissertation adapts these characteristics of Tabu Search and VNS to a Genetic Algorithm heuristic. Specifically, it uses a structured search space, that is, a spatial distribution of individuals, to generate initial populations. In addition, it concentrates the search by developing a cross-over operator that works within a spatially defined neighborhood when generating offspring. Lastly, it develops a mutation operator that is capable of introducing changes to the offspring that force it to move outside of a defined neighborhood in order to adequately explore the global search space.

Applying the Genetic Algorithm Heuristic to the *P*-median Problem

The canonical genetic algorithm, as defined by Holland (1975) and applied to the *p*-median problem by Hosage and Goodchild (1986), encoded the search space as a binary string. Dibble and Densham changed that and encoded the search space as an index of a set of nodes (1993). This change yielded improved results that were comparable to the interchange method used by Tietz and Bart (1968). The algorithm in this dissertation takes this a step further and encodes the search space in a way that preserves the spatial relationship of the nodes.

There has been very little research published regarding methods for generating the initial generation of chromosomes when using a genetic algorithm on the *p*-median problem. The approach taken by Hosage and Goodchild (1986) was to randomly generate the first generation. With the exception of one paper (Chiou & Lan, 2001), all subsequently published research in this area has taken the same approach. Similar to Chiou and Lan, the algorithm in this dissertation will take an approach that creates a

structured initial generation from a spatially distributed search space. This approach will use an algorithm to uniformly partition the search space into non-overlapping regions and then select a gene from each region to form a chromosome that will be added to the pool of chromosomes that compose the initial generation.

Hosage and Goodchild (1986) used a strictly random method for selecting individuals from a population for use in generating offspring. Subsequent to that research, several techniques have been developed that attempt to mate fitter individuals from the population with the expectation that the resulting offspring will also be fit (Bozkaya, et al., 2002; Correa, et al., 2001). The approach in this paper adapts a technique used by Laszlo and Mukherjee (2006) on the *k*-means problem where they used roulette wheel sampling to select individuals based on their scaled fitness.

In a genetic algorithm the crossover operator acts to merge the genes of the chromosomes selected for reproduction in a prescribed way to produce offspring. The canonical genetic algorithm as applied by Hosage and Goodchild (1986) splits the parents into two, creating four partial chromosomes, and then these four pieces are crossed and re-combined to create two new chromosomes, one of which is randomly discarded. This technique was shown to be inefficient in that it could produce offspring identical to the parents and could decrease diversity by reducing the number of distinct solutions in the population (Bozkaya, et al., 2002). Subsequent research sought to improve the crossover operator with more deterministic techniques as well as adapting it to alternate encoding schemes (Alp, et al., 2003; Bozkaya, et al., 2002). The mutation operator seeks to add diversity in order to more fully explore the workspace. Typically, it randomly selects a small number of genes from a potential offspring and replaces them with randomly

selected new genes. This dissertation introduces crossover and mutation operators that consider spatial distances as part of their operation. In doing so, the operators can maintain the diversity necessary to support adequate exploration of the search space and minimize the operational cost associated with exploiting promising solutions.

Research Goals

The goal of this dissertation is to examine the impact of integrating domain knowledge into a genetic algorithm as applied to the *p*-median problem. The genetic algorithm uses a method for encoding that incorporates spatial location; creates a structured initial population using domain knowledge; is biased toward fitter chromosomes when selecting mating pairs; generates offspring with a spatially sensitive crossover operator; and ensures diversity with a mutation operator that is both biased and spatially sensitive. Using published problem sets that have established "best known" solutions, the study examines solutions identified by the genetic algorithm in order to determine the accuracy, efficiency and performance characteristics of the genetic algorithm. In addition it tests the contribution of each of the algorithm's operators by systematically controlling for all the other factors.

Significance & Relevance

Using a genetic algorithm to find solutions to the *p*-median problem is not new. It was first studied in 1986 (Hosage & Goodchild, 1986) and has been the subject of several

subsequent studies with the most recent being published in 2006 (Fathali, 2006). In reading these studies it can be seen that decisions made by the researchers with regards to characteristics of the algorithm such as encoding and genetic operators has a significant impact on the efficiency of the algorithm and the accuracy of the results. These decisions are able to move the algorithm from being inferior to other metaheuristic techniques to being competitive and in some situations superior to other techniques (Mladenovi, et al., 2007) while maintaining the basic characteristics of genetic algorithms as defined by Holland (1975).

The studies to date, while significant, by no means exhaust the potential for improvement that additional research into the characteristics of genetic algorithms as applied to the *p*-median problem could bring. For example, little research has been published on what impact the starting point, or initial generation of chromosomes, has on the quality of the results. In addition, exploiting the spatial nature of the *p*-median problem to improve selection, crossover and mutation operators through the use of "neighborhoods" has not been considered in any of the published literature. This is a concept that could potentially yield significant positive results.

This dissertation works within the characteristics of a canonical genetic algorithm. It explores the components of a genetic algorithm as applied to the *p*-median problem while maintaining their simplicity and ease of implementation. It also exploits domain knowledge where possible with the goal of better understanding how the use of domain knowledge can result in an improved algorithm.

Chapter 2

Review of the Literature

This chapter provides a review of prior published research that is relevant to the dissertation topic. The focus of these papers either highlights the problem being addressed or are being used to help formulate the research. The prior published research starts with a review of the history and theoretical framework of Genetic Algorithms. Next there is a review of research of other heuristic techniques specifically as they are applied to the *p*-median problem. Finishing with a review of published research in which the authors have developed what would generally be accepted as a genetic algorithm to specifically solve the *p*-median problem. This chapter will conclude with a summary of what is known based on the published literature and how this dissertation extends that body of knowledge.

Published Research on Genetic Algorithms

John Holland first published his concepts about genetic algorithms in his book *Adaption in Natural and Artificial Systems* (1975). Holland's original goal was not to design algorithms to solve specific problems, but rather to formally study the phenomenon of adaption as it occurs in nature and to develop methods for mimicking natural selection with computer systems. Holland presented the genetic algorithm as an

abstraction of biological evolution and gave the theoretical framework for adaptation under the genetic algorithm.

Holland's influence was very important, but other researchers with different backgrounds were also involved in developing similar ideas. German researcher, Ingo Rechenberg (1973) developed the idea of the "Evolution Strategy". In the United States, Bremermann (1962) and others (Fogel, Owens, & Walsh, 1966) published their idea for what they called "Evolutionary Programming". While these ideas all had unique characteristics, they all incorporated the Darwinian concepts of mutation and selection to incrementally move toward goals. Unlike these earlier evolutionary algorithms, which focused on mutation, Holland's genetic algorithm also introduced the idea of recombination, which at that time was unique to genetic algorithms.

In 1975 one of Holland's doctoral students completed a doctoral thesis that provided a comprehensive treatment of the genetic algorithm's capabilities with regard to optimization (DeJong, 1975). There was little published research after that until the First International Conference on Genetic Algorithms was held in Pittsburgh, Pennsylvania in 1985. Subsequent to that conference, another graduate student of Holland's, David Goldberg, wrote an influential, and many consider seminal book on the subject, *Genetic Algorithms in Search, Optimization, and Machine Learning* (Goldberg, 1989).

A theory of why genetic algorithms work is explained in detail in the research published by Whitley (1994) where he examines schema theory and intrinsic parallelism. Conceptually, the theory refers to the ability of the algorithm to preserve the common sections of the solutions being evaluated that have superior fitness values. This happens when, as the algorithm processes, some sub-sets of the solution sets being evaluated

converge and together form a particular schema. The algorithm consistently disregards schemata that correspond to inferior solutions and evaluates more and more of the schemata that correspond to solutions with better fitness values.

In Holland's early research (1975) he emphasized the need for a general purpose genetic algorithm rather than domain specific implementations. However, in any actual implementation of a genetic algorithm, understanding the domain is necessary in order to make key decisions with regard to the design of the algorithm. Adaptively finding structures that perform well in a given environment is central to the concept of genetic algorithms (Whitley, 1994). If those structures are solutions to a problem and the environment is a particular domain, it is necessary to understand the domain in order to judge the "goodness" of a solution. In other words, solutions are only valid in the context of a given domain.

Published Research on Heuristic Approaches to the *P*-median Problem

A thorough survey of the literature on heuristic methods for solving the *p*-median problem was developed by Joshua Reese (2005). While this survey does a good job of annotating the existing literature it doesn't provide quantitative details on the methods or information on how the methods compare relative to each other. Fortunately, two recently published studies make up for the deficiency by providing a more detailed analysis of the heuristic approaches to solving the *p*-median problem (Alba & Dominguez, 2006; Mladenovi, et al., 2007). Mladenovi et al divide the heuristics into two groups labeled Classic Heuristics and Metaheuristics. The techniques identified as Classic Heuristics are

shown to not be competitive with the techniques identified as Metaheuristics as the problem size increases. Mladenovi et al define Metaheuristics as "a general framework to build heuristics for combinatorial and global optimization problems." The techniques Mladenovi et al identified as Metaheuristics include: Tabu search, Variable neighborhood search, Genetic algorithm, Scatter search, Simulated annealing, Heuristic concentration, Ant colony optimization, Neural networks, Decomposition, and Hybrids. Most of these techniques were applied to either the OR-Library or TSP-Library or sometimes both. In almost every case, the metaheuristic showed results that greatly exceeded the classic heuristic approaches.

While most of these techniques show the value of a heuristic approach to combinatorial problems in general and the *p*-median problem specifically, they do not have a direct influence on this dissertation. Two of the techniques do have a more direct influence (Hansen & Mladenovic, 1997; Rolland, et al., 1996). In the Tabu Search procedure developed by Rolland et al they introduce the concept of a "neighborhood" to help focus the search on promising solutions. The neighborhood is defined as the set of solutions that can be reached by either adding a single facility or dropping a single facility from the set of open facilities. As these moves are performed, tabu restrictions are used to avoid moving back to solutions that have already been considered. Tabu restrictions also enforce the neighborhood concept which allows the algorithm to incrementally move toward an optimal solution rather than introducing radical and potentially disruptive changes. Rolland et al also introduce the concept of diversification into their Tabu search algorithm. Diversification is used to escape from local optima and is implemented by using a frequency function that creates a bias against performing the

same move too often. This technique causes the algorithm to "diversify" its search into areas of the problem set that have not been investigated. Both the concept of "neighborhood" and "diversification" in search are relevant to this dissertation. In the algorithm developed in this dissertation the crossover and mutation operators implement a neighborhood concept that is used to support an incremental approach to optimization and minimize the risk of disruptive changes that may degrade the best solutions. The dissertation algorithm also implements a biased mutation operator. The operator favors selecting nodes for insertion into solutions to be evaluated that have had a lower frequency of prior use.

A Variable Neighborhood Search for the *p*-Median problem was presented by Hansen and Mladenovic (1997). In their research they also use the concept of a "neighborhood" to intensify the search on promising areas of the problem set. Neighborhoods consist of overlapping sub-sets of the problem set centered on a local optimum and increasing in size as they expand further from that local optimum. Exploration of these neighborhoods is done in two ways. The neighborhoods closest to the current solution are explored systematically with a local search until an improved solution is found. The larger neighborhoods, i.e. those far from the current solution, are explored partially by randomly selecting a solution from the neighborhood and starting a local search from there. The algorithm remains at the same solution until a better one is found and then jumps to that solution. Neighborhoods are ranked so that solutions are explored increasingly far from the current one. This ranking allows the search to intensify around and diversify from the current solution through an intrinsic "shaking" process. The level of shaking is set through an execution parameter. Hansen and Mladenovic's

research is relevant to this dissertation because they hypothesize that the reason that Variable Neighborhood Search algorithms work is because "all good *p*-median solutions are 'relatively' close to each other with respect to distance". Their published research supports that hypothesis. Their research is important because the algorithm developed in this dissertation implements genetic operators designed to take advantage of this localization of good solutions.

Published Research on Genetic Algorithms for the *P*-median Problem

In the research by Hosage and Goodchild (1986), they develop the first genetic algorithm published in the literature that provides a solution to the *p*-median problem. Their algorithm conformed closely to the canonical genetic algorithm developed by Holland (1975). In their algorithm, Hosage and Goodchild encode a solution as a string of *m* binary digits which they referred to as genes. The allele of each binary digit is set to 1 if it represents a facility and 0 if it represents a demand node. In addition to the crossover and mutation operators, Hosage and Goodchild incorporate an inversion operator. The inversion operator flips the alleles of selected chromosomes in an attempt to introduce additional genetic diversity. However, as the percentage of chromosomes selected for inversion increases, the tendency of the algorithm to perform similarly to random search also increases. Subsequent research by Goldberg (1989) cast doubt on the value of an inversion operator given its significant computational cost. Hosage and Goodchild used a pre-determined number of generations as a stopping criterion, rather than a solution convergence because of their concern about the possibility of converging on a local

optimum rather than a global one. While premature convergence is a concern addressed in subsequent research, the use of pre-defined stopping points resulted in consistent results. Hosage and Goodchild's algorithm showed poor computational results. In their conclusion, Hosage and Goodchild acknowledged the poor computational results but asserted that the value of their approach was its general applicability to a large set of problems rather than its computation efficiency. Hosage and Goodchild's primary contribution was being the first to develop a working genetic algorithm for the *p*-median problem.

In Dibble and Densham (1993), each chromosome has exactly *p* genes, and each gene represents a facility index. This appears to be a better encoding technique than the binary string approach used by Hosage and Goodchild (1986). Dibble and Densham used conventional genetic operators: selection, cross-over and mutation, but no inversion operator. Reported results are similar to Interchange local search, but with considerably longer processing time. Dibble and Densham's primary contributions were an improved method of encoding the problem onto the chromosome by using index pointers and a head-to-head comparison with another heuristic for the *p*-median problem. The algorithm developed in this dissertation further refines the encoding technique and represents each gene within a chromosome as a multi-dimensional vector containing the coordinates of a candidate median. This technique is very similar to the one used by Laszlo and Mukherjee (2007) in their work on a genetic algorithm for the *k*-means problem.

In Estivill-Castro and Torres-Velazquez (1999), a mutation operator is introduced that is based on a hill-climber algorithm. Their mutation operator randomly selected chromosomes for improvement using a hill-climbing technique and then reintroduced the

chromosome back to the population. Estivill-Castro and Torres-Velazquez also experiment with various crossover operators but ultimately conclude that the increased computational complexity offset any gains achieved by earlier convergence. While no operational data is presented, the authors claim that the algorithm outperforms tabu search and simulated annealing algorithms applied to similar data sets. By extending the functionality of the mutation operator, Estivill-Castro and Torres-Velazquez show that it can be beneficial to have potential solutions survive from generation to generation. In their case they did that through the mutation operator. In this dissertation algorithm a "hero" chromosome is introduced that represents the best solution in the current generation and is immune to the cross-over and mutation operators and will be passed intact to the next generation through the replacement operator.

The primary focus of a study by Chiou and Lan (2001) is clustering. It has relevance to this dissertation because it develops a method referred to as the Cluster Seed Points Method (CSPM) for developing the first generation in a genetic algorithm which in turn is used on the *p*-median problem. The operators used in the Chiou and Lan genetic algorithm were very standard but their use of CSPM for generating the initial population of chromosomes showed improvement over techniques that randomly generated the initial population and was the first published research that used a directed approach rather than a random approach. CSPM designs initial populations by manually selecting "seeds" from the search space for each initial population. This method, using structured initial populations, showed good results however it severely limits the dynamism of the algorithm. In addition, the experiment was applied only to a small search space. Chiou and Lan stated in their conclusion that the CSPM method would probably not scale well

to larger search spaces. In other related research, Arthur and Vassilvitsakii (2007) used seeding in a *k*-means algorithm. While not directly applicable to the *p*-median problem, it does provide mathematical support for efficacy of seeding for combinatorial problems. This dissertation elaborates on the findings in these papers in support of developing a seeding technique that provides a good starting point for the genetic algorithm rather than relying on random selection.

In a study by Correa et al (2001) a genetic algorithm for the capacitated *p*-median problem is presented. This is a slightly different combinatorial problem than the *p*-median problem in that servicing facilities have a limited capacity so the algorithm must consider both distance and availability when calculating cost. In a genetic algorithm, this primarily affects the fitness function. The chromosome encoding and the operators are the same for either problem and as such, this research is applicable to the research for this dissertation. The research by Correa et al is unique in two aspects. First, they use a ranking based selection operator. Specifically, prior to selection they rank chromosomes in the population from most fit to least fit. They then apply a selection formula that is biased toward chromosomes that appear early in the list thus tending toward selecting more fit chromosomes. This dissertation algorithm uses a conceptually similar technique, however instead of ranking by fitness; it uses a scaled fitness function and "roulette-wheel" selection which gives the fitter solutions more likelihood of selection. The second unique characteristic of the Correa et al algorithm is something they refer to as a hyper-mutation operator. The hyper-mutation operator randomly selects a small percentage of chromosomes and tries to improve their fitness by evaluating every feasible median not currently represented in the chromosome. This is computationally expensive and while

Correa et al only test it on relatively small sets of data, it seems likely that its cost would out-weigh its benefit as the size of the data set grew. It also seems to negate the value of the mutation operator, which is to encourage exploration over exploitation.

In a more recent study, Alp et al (2003) developed a fast genetic algorithm with good results. Though the algorithm they present is not a genetic algorithm in the strictest sense, it is an evolutionary algorithm and contains many of the elements typically found in a genetic algorithm. Their crossover operator uses a greedy drop procedure to generate new chromosomes from chromosomes randomly selected from the current population. In this procedure, first the chromosomes of parents are merged to produce an infeasible solution with m genes where $m > p$. Then the gene whose dropping produces the best fitness function is dropped. This is repeated until number of genes reaches p. This research shows the value of directed crossover and replacement operators. The algorithm generated in this dissertation further explores improved crossover and replacement operators by experimenting with operators that take advantage of the spatial nature of the p-median problem. Alp et al do not use a mutation operator in their algorithm. They claim that when they introduced a basic mutation operator, it did not improve the solution; however no data was provided to support the claim. One final aspect of the Alp et al algorithm is its stopping criteria. Rather than simple stopping after a pre-defined number of generations their algorithm stopped after the best (most fit) solution did not change after $\lceil n\sqrt{p} \rceil$ successive children failed to improve it. This appears to be an improvement over previously published methods that simply stopped after a fixed number of generations; however, it isn't clear that it is an improvement over algorithms that use convergence for a set number of iterations as a stopping criterion.

Alp et al (2003) also perform a fairly detailed comparison of their algorithm with other heuristics for the *p*-median problem using the OR Library. A summary of the comparison is that the Alp et al algorithm performs as well as or better than the other algorithms which include a simulated annealing heuristic and a gamma heuristic. This study shows that while a basic genetic algorithm cannot compete with more recent meta-heuristics in solving the *p*-median problem, it is subject to improvement with some modifications that maintain the simplicity and ease of implementation that are characteristic of genetic algorithms.

In the most recent publication that examines the application of what would be strictly defined as a genetic algorithm to the *p*-median problem, Bozkaya et al (2002) present a new algorithm. Their algorithm retains all the typical characteristics of a genetic algorithm and outperforms previously published genetic algorithms, and the Tietz and Bart (1968) interchange algorithm, in terms of accuracy and processing times. The components of the algorithm developed by Bozkaya et al are not necessarily unique to their work. What is unique is their combination of previously examined components into a new algorithm that draws on promising techniques to form what can be considered a "best-of-breed" genetic algorithm. Their contribution to the body of knowledge is showing that while the basic genetic algorithm for the *p*-median problem developed by Hosage and Goodchild (1986) is not competitive with other techniques, a well designed algorithm can be, while still maintaining all the characteristics of the canonical genetic algorithm. There is, however, one aspect of their work that is unique and directly applicable to this dissertation. They use a formula for setting the number of solutions or chromosomes that will make up the population *P* of a generation. The formula they

introduce is given as $P = \left\lceil \left(\frac{1}{p}\right)\left(\frac{\ln P_0}{\ln\frac{n-1}{n}}\right)\right\rceil$ and where P_0 represents the probability of not including a node in the initial population. This technique shows significant improvement over other methods and is adopted in the algorithm developed for this dissertation.

Chapter 3

Methodology

This dissertation examines the impact of integrating domain knowledge into a genetic algorithm as applied to the *p*-median problem. To do that, a new domain aware genetic algorithm (DAGA) has been developed. In addition, a set of tests are carried out that examine both the overall efficacy of this algorithm as well contributions of individual components of this algorithm. Both the algorithm and tests are described in more detail in the following sub-sections.

Algorithm Design

The DAGA uses the same general structure and genetic operators as the canonical genetic algorithm defined by Holland (1975). This dissertation uses Holland's theoretical framework and presents a domain aware genetic algorithm by developing the following: a scheme for encoding the problem set into genes, alleles, and chromosomes; a technique for generating the first generation of chromosomes; a technique for selecting chromosomes from the current generation for use in generating chromosomes for the next generation; a technique for combining chromosome pairs to create offspring chromosomes; and a technique for mutating new chromosomes. A description of the approach to each of these components is provided in the following paragraphs.

Encoding

The DAGA uses an object-oriented approach to encoding the problem set. A Node class is generated and an instance of this class is generated for each vector in the problem set. In the *p*-median problem, each vector represents the point coordinates of a specific location. The set of all locations is represented in the problem formulation as n and n_i represents a specific location within the problem set. The Node class acts as a generalization of the Gene class. An instance of the Gene class is generated for each vector within n that is part of a feasible solution set. In the *p*-median problem, each vector within a solution set represents a median. The set of all medians within a feasible solution is represented by p and p_i represents a specific median within a feasible solution set. A Chromosome class has been developed and an instance of this class is generated for each feasible solution set within the set of feasible solutions that represents a generation during the algorithm's execution. The Chromosome class has a composite association with the Gene class whereby an instance of the Chromosome class is made up of p instances of the Gene class. A Generation class has been developed and an instance of this class is generated for each set of chromosomes that constitute a generation. The Generation class has a composite association with the Chromosome class whereby an instance of the Generation class is made up of P instances of the Chromosome class. A UML diagram of these classes and their relationships is given in Figure 2. **UML Diagram**.

The Node class attributes include an attribute containing the location vector, an attribute containing a count of the number of times an instance of the location is being used in the current generation of chromosomes, and an attribute containing a count of the total number of times the location has been used in any chromosome. In addition to the

attributes inherited from the Node class, the Gene class attributes include a unique identifier, a Boolean value indicating whether this instance has been selected for crossover operations, and a Boolean value indicating whether it has been selected for mutation operations. The attributes of the Chromosome class include a unique identifier and a value indicating the calculated fitness of the solution set. Though not shown, each class will also have the operators necessary to implement the classes as part of the algorithm.

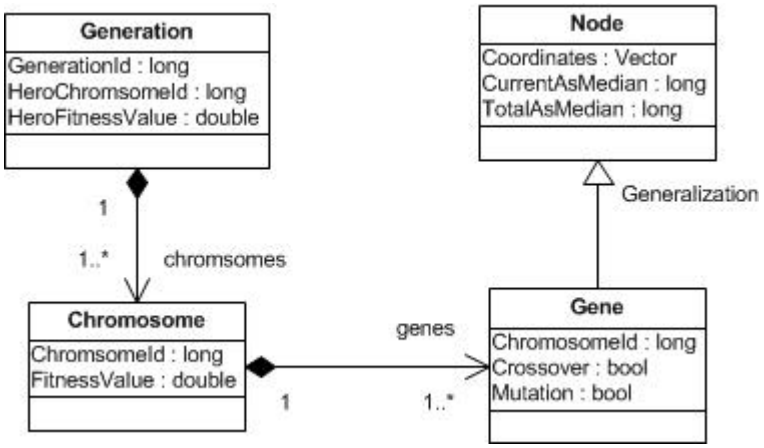


Figure 2. UML Diagram

This object-oriented approach to encoding the problem is primarily an implementation issue. From a research perspective, it is not significantly different than the technique used by Dibble & Densham (1993). Their encoding technique is based on *p* length chromosomes where the alleles of the genes correspond to the indices of selected medians. Dibble and Densham showed that their encoding technique was significantly superior to the binary string representation technique first used by Hosage and Goodchild (1986). It is expected that this objected-oriented technique will be equally as effective.

Run-time parameters

Some of the characteristics of the DAGA can be controlled at the time of execution by setting parameter values. Specifically, there are six parameters that must be set at run-time that impact the operation of the algorithm and have an impact on the results. Those parameters are Population Size parameter, the Stopping Criteria parameter, the Selection Threshold parameter, the Crossover Threshold parameter, the Chromosome Mutation Rate parameter, and the Gene Mutation Rate parameter.

A formula for determining the population size *P* was presented by Bozkaya et al (2002) and sought to include as many distinct nodes in the initial population as possible. The DAGA adopts this formula for setting the initial population size. The formula was given as $P = \left\lceil \left(\frac{1}{p}\right)\left(\frac{\ln P_0}{\ln\frac{n-1}{n}}\right)\right\rceil$ where *P* represents the number of chromosomes in the initial population and P_0 is the Population Size parameter and represents the probability of not including a node in the initial population. Because it is likely that the probability of a node being introduced into the population by mutation is small, the probability of a node missing from the initial population should be correspondingly small.

The two approaches commonly used to decide when to terminate a genetic algorithm are setting a defined number of iterations or generations, and setting a number of iterations in which the best solution does not change. The DAGA takes the later approach and assigns a value to the Stopping Criteria parameter which is used to determine when to terminate the algorithm. If the number of successive generations in which the fittest chromosome in the population has not changed equals the Stopping Criteria parameter, the algorithm assumes it has found an optimal or near optimal solution and terminates.

The Selection Threshold parameter represents the percentage of chromosomes in a parent generation that will be selected to act as parent chromosomes in the crossover operation. The Crossover Threshold parameter represents the percentage of genes in a parent chromosome that will be swapped with the genes from the paired parent chromosome to produce offspring. Both of these parameters would typically be set at around 50% however they are experimented with to determine how differing thresholds affect the algorithms operation.

The Chromosome Mutation Rate parameter represents the percentage of chromosomes in a child generation that are selected for mutation prior to being used as the next generation. The Gene Mutation Rate parameter represents the percentage of genes in a chromosome selected for mutation that will be subjected to mutation. These numbers must work in concert and be set low enough to avoid disrupting promising solutions and high enough to ensure that all nodes are considered and to encourage moving beyond local optima.

Initial populations

A random approach to creating the initial population of chromosomes has been used by most published research on using a genetic algorithm to solve the *p*-median problem to date. However, several studies on other problems show that the initial population can have a significant impact on the effectiveness of an algorithm (Arthur & Vassilvitskii, 2007; Chiou & Lan, 2001; Laszlo & Mukherjee, 2006). The algorithm developed in this study uses a technique that uniformly partitions the search space into non-overlapping regions and then generates the initial population by randomly selecting a single gene from each region for each chromosome in the first generation.

A *PR KD-Tree* approach is used to partition the nodes within the search space. Given a *K* dimensional search space containing *N* nodes, *p* non-overlapping regions will be generated ($R_{1\text{-}p}$) where *p* represents the number of medians defined in the given *p*-median problem. To create the regions, the region containing the greatest number of nodes is selected and divided to create two new regions. This process continues until *p* regions have been created with at least one node in each region. To divide a selected region, a dimension, *K*, is cyclically selected and a dividing point *MK* is selected along the axis represented by *K*. The dividing point is selected by identifying the point on the *K* axis that is the median of the node values in the region in the K^{th} dimension. All nodes with a value in the K^{th} dimension less than *MK* are added to one node and all nodes with a value in the K^{th} dimension greater than or equal to *MK* are added into the other. These two new regions will replace the original region. When complete, this technique results in the search space being divided up into *p* non-overlapping regions roughly representing the density of the nodes within the search space.

To generate the initial generation of chromosomes, individual chromosomes are created by selecting one node from each region to act as a gene in the chromosome being built. This process continues until the percentage of nodes represented as a gene in one or more chromosomes exceeds a given threshold parameter. When the given threshold has been exceeded, the chromosomes that have been created will be the initial generation.

Selection

To create the next generation of chromosomes, a genetic algorithm must select pairs of chromosomes from the current population to be used to create chromosomes to be used in the next generation. The DAGA uses a two-step method for selection. In the

first step, the fittest chromosome, based on the fitness function, is isolated and protected from change by the crossover or mutation operators. When the next generation is formed, this chromosome will be added unchanged to the next generation. Of the remaining chromosomes, a fitness proportionate, or roulette wheel, technique is used to select mating pairs. In this technique, a random number is generated between 0 and the sum of the reciprocal of the fitness value of all chromosomes in the population excluding the "hero" chromosome. The equation for this is given as $r = Random\left(0, \sum_{i=1}^{P} \frac{1}{f(i)}\right)$ whereas *P* is the number of chromosomes in the population and *f()* is the fitness function. Using *r* as a threshold value, incrementally sum the reciprocal of the fitness function value for each chromosome until the total equals or exceeds *r*. The chromosome that causes the total to equal or exceed *r* is selected. Using this selection technique, two chromosomes are selected from the current generation to act as a mating pair. If the pair has not previously been selected, it is added into a mating pair pool. This process repeats until enough mating pairs have been selected to create *P* offspring to be used for the next generation.

Theoretically, more fit parents will result in more fit children. This selection technique is biased toward fitter chromosomes but does not preclude the possibility of selection of less fit individuals to help ensure adequate genetic diversity.

Crossover

The crossover operator's primary function is to allow the algorithm to explore or "walk" the search space. It does that by creating new chromosomes made up of genes inherited from parent chromosomes. There are a wide variety of techniques, or operators, for selecting genes for crossover described in the literature. This dissertation experiments

with two different operators, both of which will take advantage of the spatial nature of the *p*-median problem and incorporate gene location into the process.

The canonical approach to the crossover operator is to simply split the parent chromosomes in half and then reform the halves into one or two child chromosomes. The simplicity of this technique can result in significant operational efficiencies. It does however leave much room for improvement in the efficiency of the search. The first technique to be explored in this dissertation seeks to improve search efficiency by working with individual genes and making use of a "nearest neighbor search" as defined by Samet (2006). The technique is shown in Figure 3. **Crossover Technique 1** and described in the following Steps:

Step 1. Randomly select one of the chromosomes from the mating pair and consider it the Primary Parent Chromosome C_1. Consider the other chromosome in the pair as the Secondary Parent Chromosome C_2.

Step 2. Make a copy of the Primary Parent Chromosome and consider it the Primary Offspring Chromosome C'_1. Make a copy of the Secondary Parent Chromosome and consider it the Secondary Offspring Chromosome C'_2.

Step 3. Randomly select a gene p_1 from C_1. Find the Location L in C_2 that corresponds to the location coordinates of p_1.

Step 4. Using a "nearest neighbor search" find the gene p_2 in C_2 that is closest to L.

Step 5. In the Primary Offspring Chromosome C'_1 replace gene p_1 with gene p_2 from the Secondary Parent Chromosome C_2. In the Secondary Offspring Chromosome C'_2 replace gene p_2 with gene p_1 from the Primary Parent Chromosome C_1.

Step 6. If the number of genes replaced in the Offspring Chromosomes is less than the value of the Crossover Threshold parameter given at run-time, return to Step 3 and process through the remaining steps again.

Step 7. Add C'_1 and C'_2 to candidate pool for the next generation chromosomes.

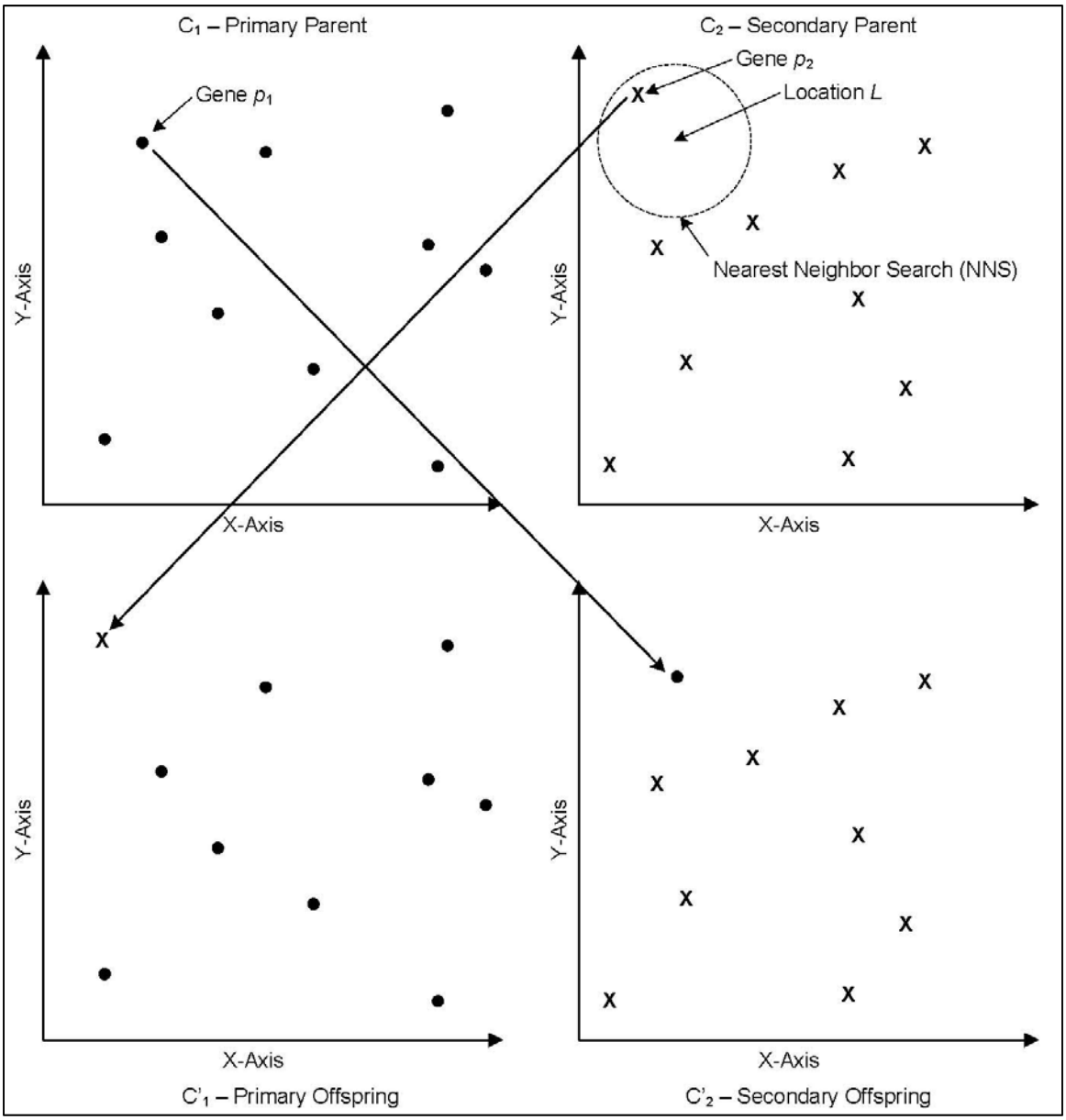


Figure 3. Crossover Technique 1

This technique tests the concept that the additional computational expense required by the crossover operator is overcome by producing a more efficient walk through the search space.

The second technique seeks the middle ground between the computational efficiency of the canonical crossover operator and the search efficiency of the first crossover operator described above. In this technique, the operator splits the chromosomes based on the location of the genes on a selected axis and recombines them to form child chromosomes with the same number of genes as their parents. The technique is illustrated in Figure 4. **Crossover Technique 2** and described in the following Steps:

Step 1. Randomly select one of the dimensions that make up the search space *d*. Then identify a cutoff value (d_c) that equals *p* multiplied by the Crossover Threshold parameter given as a run-time parameter.

Step 2. Randomly select one of the chromosomes from the mating pair and consider it the Primary Parent Chromosome C_1. Consider the other chromosome in the pair as the Secondary Parent Chromosome C_2.

Step 3. In the Primary Parent Chromosome C_1, find the unselected gene with the highest value on the *d* axis (p_{max}) and copy that gene to Primary Offspring Chromosome C'_1. Continue this process until the count of genes copied from C_1 to C'_1 equals or exceeds the cutoff value d_c.

Step 4. Copy all remaining unselected genes in the Primary Parent Chromosome C_1 to the Secondary Offspring Chromosome C'_2.

Step 5. In the Secondary Parent Chromosome C_2, find the unselected gene with the highest value on the *d* axis (p_{max}) and copy that gene to Secondary Offspring Chromosome C'_2. Continue this process until the count of genes copied from C_2 to C'_2 equals or exceeds the cutoff value d_c.

Step 6. Copy all remaining unselected genes in the Secondary Parent Chromosome C_2 to the Primary Offspring Chromosome C'_1.

Step 7. Add C'_1 and C'_2 to candidate pool for the next generation chromosomes.

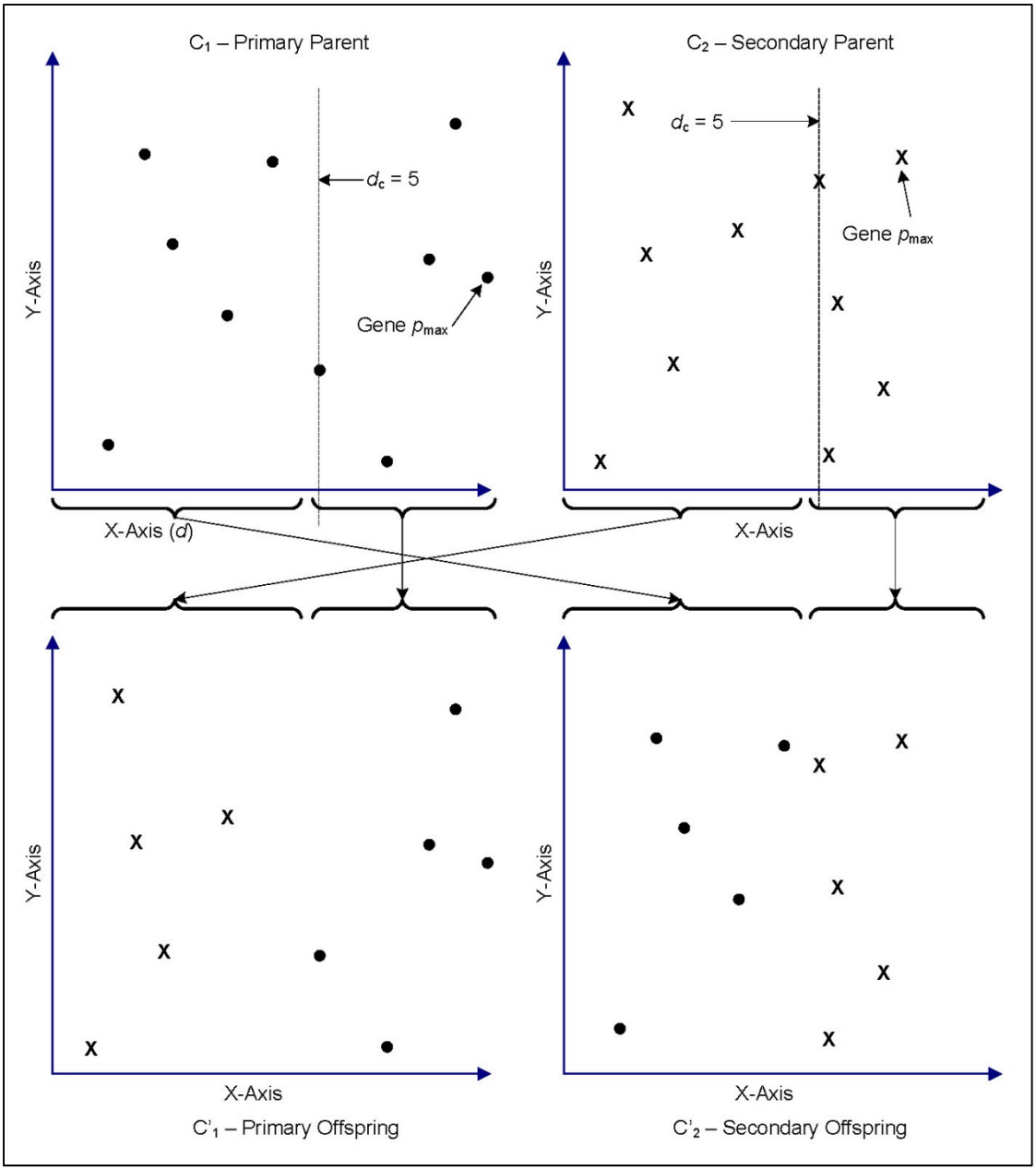


Figure 4. Crossover Technique 2

This technique seeks to determine whether a method that is domain aware but less computationally intensive than the first technique tested can yield overall improved results.

Whichever technique proves superior, the concept is that any additional computational expense required will be overcome by producing a more efficient walk through the search space. This is accomplished by allowing the search to exploit crossover operations that have a higher likelihood of increasing the fitness of the child chromosome.

Mutation

The purpose of the mutation operator in a genetic algorithm is to introduce diversity into the search in order to encourage a thorough evaluation of the search space. The most common technique described in the literature is to simply randomly select genes from the potential offspring and replace those genes with others. The DAGA uses a more deterministic technique. It is biased towards selecting nodes for insertion into offspring chromosome candidates that have been used fewer times as genes or medians. For example, a node that has been used once as a median in any chromosome in all prior generations will be twice as likely to be selected as one that has been used twice. This is done by using a proportionate or "roulette wheel" selection technique. In this technique, a random number is generated between 0 and the sum of the reciprocal of the usage count of all nodes in the problem set. The equation for this is given as $r = Random\left(0, \sum_{i=1}^{n} \frac{1}{u(i)+1}\right)$ whereas n is the number of nodes in the problem set and $u()$ is the prior use function. Using r as a threshold value, incrementally sum the reciprocal of the prior use function plus one for each node in the problem set until the total equals or exceeds r. The node that causes the total to equal or exceed r is selected.

In addition to a selection bias, the DAGA mutation operator considers gene location during the substitution process. Specifically, the gene being inserted will replace the gene that is located closest to it.

The purpose of using a biased selection technique is to increase the probability that a node within the problem set will be evaluated as a median. The purpose of replacing genes with new genes located nearby is to facilitate the continual improvement of the solution by reducing the risk of large disruptive changes to the chromosome.

The DAGA mutation operator is illustrated in Figure 5. **Mutation Operator** and is described in more detail in the following steps:

Step 1. Select a node p_m from the set of all nodes n in the problem set using a "Roulette Wheel" selection technique that is biased towards nodes with lower prior use counts.

Step 2. Randomly select a chromosome C'_m from the offspring candidate pool C'.

Step 3. Insert the selected node p_m into the select chromosome C'_m.

Step 4. Using a "nearest neighbor" search technique, locate the gene p_r located nearest to the inserted gene p_m.

Step 5. Remove p_r from the selected chromosome C'_m.

Step 6. If the total number of chromosome selected for mutation is less than the value derived from the Mutation Rate parameter (Mutation Rate multiplied by population size), return to Step 1 and process through the all the steps again.

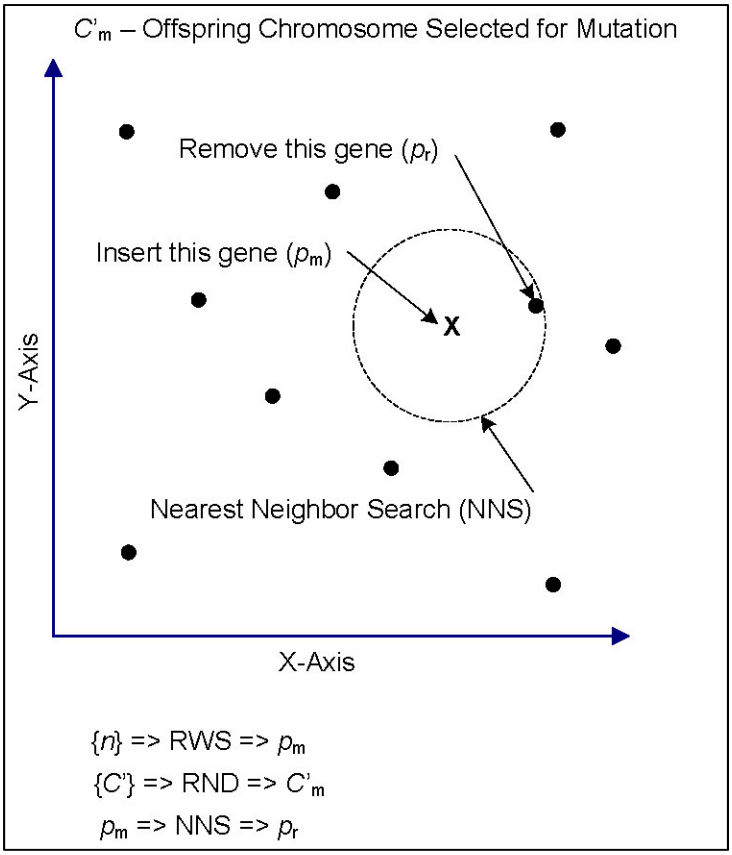


Figure 5. Mutation Operator

Computational Study

This dissertation conducts experiments to determine whether DAGA can find solutions that are as good or nearly as good at the solutions found by other genetic algorithms published in the literature. It does that by running DAGA using selected datasets from the TSP Library (Reinelt, 1991). The TSP Library was originally developed as a set of problem sets for the Travelling Sales Person problem however it has been used extensively in literature as a problem set for the *p*-median problem (Alba & Dominguez, 2006; Avella, Sassano, & Vasil'ev, 2007; Beltran, Tadonki, & Vial, 2006; García-López, Melián-Batista, Moreno-Pérez, & Moreno-Vega, 2002; Hansen & Mladenovic, 1997,

2007; Hansen, Mladenović, & Perez-Britos, 2001; Resende & Werneck, 2004). The problem sets are made up of sets of two dimensional Cartesian coordinates with sets ranging in size from 29 to 13509 points. The three problem sets from the TSP Library that have been most widely used in the literature for the *p*-median problem are fl1400, pcb3038, and rl5934. A complete list of the problems sets used, their best known solutions, and the source of those solutions are shown in Table 1. **fl1400 Problem Set**, Table 2. **pcb3038 Problem Set,** and Table 3. **rl5934 Problem Set.**

Table 1. fl1400 Problem Set

n	p	Best Known	Study
1400	10	101,248.13	Hansen (1997)
1400	20	57,856.32	Hansen (1997)
1400	30	44,013.02	Alba (2006)
1400	40	35,002.02	Alba (2006)
1400	50	29,089.71	Alba (2006)
1400	60	25,160.40	Alba (2006)
1400	70	22,125.46	Alba (2006)
1400	80	19,870.28	Hansen (2007)
1400	90	17,987.91	Alba (2006)
1400	100	16,551.20	Hansen (1997)

Table 2. pcb3038 Problem Set

n	p	Best Known	Study
3038	10	1,213,082.03	Resende (2004)
3038	20	840,844.53	Resende (2004)
3038	30	677,436.66	Resende (2004)
3038	40	571,887.75	Resende (2004)
3038	50	507,655.19	Hansen (2001)
3038	60	460,797.55	Resende (2004)
3038	70	426,153.31	Resende (2004)
3038	80	397,585.89	Resende (2004)
3038	90	373,488.82	Resende (2004)
3038	100	352,755.13	Resende (2004)

Table 3. rl5934 Problem Set.

n	p	Best Known	Study
5934	10	9,794,951.00	Hansen (2001)
5934	20	6,729,282.50	Hansen (2001)
5934	30	5,405,661.50	Hansen (2001)
5934	40	4,574,374.00	Hansen (2001)
5934	50	4,053,917.75	Hansen (2001)
5934	60	3,655,898.75	Hansen (2001)
5934	70	3,353,885.00	Hansen (2001)
5934	80	3,104,877.75	Hansen (2001)
5934	90	2,903,895.25	Hansen (2001)
5934	100	2,733,817.25	Hansen (2001)

A summary of the results with descriptive analysis are presented in Chapter 4 of this dissertation. The complete results from all of the runs are shown in Appendix B.

Experiments with run-time parameters

The algorithm allows for some parameters to be set that impact various aspects of the operation of the algorithm. Those parameters include: a value that the probability of not including a node in the initial population. This value indirectly impacts the number of chromosomes that make up a generation. A value that determines what percentage of chromosomes from the parent generation are selected to be used in the crossover operation to generate offspring for the next generation. A value that determines what percentage of genes from a chromosome undergoing crossover should be selected from each parent chromosome. A value that determines what percentage of chromosomes in a child generation are selected for mutation. A value that determines what percentage of genes in a chromosome undergoing mutation will be replaced. Finally, a stopping criterion is set. The stopping criterion determines how many generations must pass without the best fitness value improving in order for the algorithm to terminate.

These values were tested in various combinations and a single overall best configuration is determined. This configuration is then used during all instances of the testing for both the accuracy and efficiency of the algorithm.

Experiments on the effectiveness of the Domain Aware Genetic Algorithm

A test plan was used to study the effectiveness of the DAGA. The test plan applies variations of the algorithm to the selected problem sets and median counts. The first variation used the Crossover Technique 1 and the next variation used Crossover Technique 2. The algorithm was run ten times for ten medians in each of the selected problem sets. From the ten runs, the run with the lowest fitness value was identified as the lower bound. The run with the highest fitness value was identified as the upper bound, and an average of all ten runs was also calculated. For the lower bound result, upper bound result and average result, an error rate was calculated by subtracting the result from the best known solution found in the published literature and then dividing the result by that best known solution. This error rate was used determine the normalized deviation from the best known solution. Finally, the gap between the lower bound error rate and the upper bound error rate is calculated to determine the consistency of the algorithm.

Experiments on the efficiency of the algorithm

In addition to testing the effectiveness of DAGA the efficiency of the algorithm was also tested. The efficiency was measured by tracking the fitness values for each generation as it evolved toward an optimal solution. The quicker, in terms of the number of generations, it improved from its initial position to a good and then optimal or near optimal solution, the more efficient the algorithm can be considered.

The results produced from the runs against the fl1400 problem set described above were further analyzed and the line graphs were created to illustrate the analysis. For each value of *p* in the test set a line graph was created that tracked two test runs representing the run that produced the lower bound value and the run that produce the upper bound value. In each graph the x-axis represents generations and the y-axis represents the deviation of the fitness value for the given generation, expressed as an error rate, from the best known solution. Each graph was constrained to the first 2500 generations to provide a common basis of comparison between the lower and upper band values as well as the different values of *p*. By converting raw fitness scores into error rates, a consistent basis for comparison is provided across all test instances. This allows some determination to be made about how variations in the algorithm impact its ability to efficiently move to an optimal or near-optimal solution.

Experiments on specific operators of the algorithm

In addition to testing the effectiveness and efficiency of DAGA, experiments were conducted to determine what impact, if any, individual operators used by DAGA had on the overall performance of the algorithm. Specifically the impact of a structured Initial Generation, a location aware Crossover Operator, and a location aware Mutation Operator, were analyzed. In each case the operator being tested was replaced with an operator that acted randomly. Specifically, when the structured initial generation operator was tested, it was replaced with an operator that randomly selected nodes to create the chromosomes for the initial generation. When the crossover operator was being tested it was replaced with an operator that randomly selected nodes from the parent chromosomes for crossover. When the mutation operator was being tested it was replaced

with an operator that randomly selected chromosomes in the candidate generation for mutation and randomly selected genes within the selected chromosomes for mutation. These modified algorithms were each run ten times for *p* values 10 through 100 in the fl1400 problem set. The lower and upper bound results were compared with the lower and upper bound results from DAGA and the Best Known results from literature. In addition, the results were graphed to compare the efficiency of the modified algorithms as compared to DAGA.

Chapter 4

Results

The results of DAGA runs are agregated and presented in a series of tables and figures in this chapter. Detailed run results are listed in Appendix B. For analysis purposes, the algorithm was run 900 times in total to test each problem set and median count combination 10 times each. The execution time of the algorithm was not considered to be applicable to the goal of the dissertation so that statistic was not collected. Prior to the analysis runs, the algorithm was run approximately 100 times with varing problem sets in order to calibrate the runtime parameters. Based on those calibration runs, the runtime parameters determined to give the best overall results were selected and are presented in the following section. Two location-aware crossover operators were analyzed to determine which provided a consistently better solution. As a result of that analysis, crossover operator 1 described in Figure 3. **Crossover Technique 1**, was selected for further analysis. It was used to analyze the efficiency of the algorithm and in the analysis of the selected components of the algorithm.

Runtime parameters

Five runtime paramaters were used in the algorithm. They are shown, along with their selected values, in Table 4. **Runtime Parameters and Selected Values**. The first runtime parameter is labeled Pnot. It was used to determine the initial population size.

The formula for determining the initial population size is $P = \left\lceil \left(\frac{1}{p}\right)\left(\frac{\ln P_0}{\ln \frac{n-1}{n}}\right) \right\rceil$ where P represents the number of chromosomes in the initial population and P_0, labeled Pnot, is the Population Size parameter that represents the probability of not including a node in the initial population. For the purposes of analysis, the probability of a node not being selected for the initial population was set at 5%. Thus, for a given problem set the initial population size is set so that 95% of the nodes are included in the initial population. Given that the only way for a node to be introduced into the population other than as part of the initial population is through the mutation operator, and the mutation rate is typically set low, a population size that was inclusive of a large subset of the available nodes was desirable.

The next runtime parameter used was the Stopping Criteria parameter labeled as Stopping_Criteria. This parameter was used to determine when to stop the algorithm. If the number of successive generations in which the fittest chromosome in the population does not change equals the Stopping Criteria parameter, the algorithm assumes it has found an optimal or near optimal solution and terminates. For the purposes of analysis the value of this parameter was set at 2500.

The Selection Threshold parameter, labeled Selection_Threshold, represents the percentage of chromosomes in a parent generation that will be selected to act as parent chromosomes in the crossover operation. The unselected chromosomes are passed unaltered to the candidate generation. The Crossover Threshold parameter, labeled Crossover_Threshold, is used in conjunction with the Selection Threshold parameter and represents the percentage of genes in a selected parent chromosome that are swapped with the genes from the paired parent chromosome to be passed to the candidate

generation. For the purposes of analysis the Crossover Threshold parameter was set at 50% and the Selection Threshold parameter was set at 75%.

The final two runtime parameters used are the Chromosome Mutation Rate, labeled ChromMutationRate, and the Gene Mutation Rate, labeled GeneMutation_Rate. The Chromosome Mutation Rate parameter represents the percentage of chromosomes in a candidate generation, after the selection and crossover operators have been applied, that are selected for mutation prior to being used as the next generation. The Gene Mutation Rate parameter represents the percentage of genes in a chromosome selected for mutation that will be subjected to mutation. These numbers must work in concert and be set low enough to avoid disrupting promising solutions and high enough to ensure that all nodes are considered and to encourage moving beyond local optima. For the purpose of analysis, these values were both set at 10%.

Table 4. Runtime Parameters and Selected Values

Pnot	0.05
Stopping_Criteria	2500
Crossover_Threshold	0.5
ChromMutation_Rate	0.1
GeneMutation_Rate	0.1
Selection_Threshold	0.75

Summary of Results Using Crossover Technique 1

Using Crossover Technique 1, illustrated in Figure 3, and problem set fl1400 (Reinelt, 1991) consisting of 1400 nodes expressed as two dimensional cartisian coordinates, the algorithm was run 10 times each for median values 10, 20, 30, 40, 50, 60, 70, 80, and 100. For each median value the run that produced the best (lowest) fitness

value was selected and identified as the lower bound. The result that produced the worst fitness function (highest) was selected and identified as the Upper Bound. The average of all runs for each median value was also calculated and identified as the average for the respective median value. Next, a Gap value was calculated that represented the percentage deviation between the lower bound value and the upper bound value. Finally, an Error Rate was calculated for both the lower bound and upper bound values that represented the deviation of the value from the best known solution. Table 5. **Summary of Results Using fl1400 Problem Set and Crossover Technique 1** shows an aggregation of the runs and the calculated values.

Table 5. Summary of Results Using fl1400 Problem Set and Crossover Technique 1

n	p	Best Known	LBOUND	ERR	UBOUND	ERR	Gap	Average	ERR
1400	10	101,248.13	101,248.57	0.00	102,711.90	0.01	0.01	102,148.43	0.01
1400	20	57,856.32	58,859.55	0.02	60,449.35	0.04	0.03	59,600.23	0.03
1400	30	44,013.02	45,404.13	0.03	47,729.94	0.08	0.05	46,477.33	0.06
1400	40	35,002.02	36,514.57	0.04	37,741.65	0.08	0.04	37,094.65	0.06
1400	50	29,089.71	30,240.72	0.04	31,262.88	0.07	0.04	30,883.37	0.06
1400	60	25,160.40	26,620.11	0.06	27,682.85	0.10	0.04	27,204.38	0.08
1400	70	22,125.46	23,412.64	0.06	24,869.24	0.12	0.07	24,034.39	0.09
1400	80	19,870.28	20,958.67	0.05	22,280.45	0.12	0.07	21,664.30	0.09
1400	90	17,987.91	19,085.52	0.06	20,025.31	0.11	0.05	19,528.79	0.09
1400	100	16,551.20	17,580.43	0.06	18,423.08	0.11	0.05	18,129.04	0.10

The results show that for 10 medians the lower bound solution was as good as the best known solution and the upper bound solution within 1% of the best known solution. The average of all runs for 10 medians was also within 1% of the best known solution and the gap between the upper and lower bounds was no more than 1%. However, as the number of medians increases from 10 to 100 the deviation from the best known solution

for the upper and lower bounds increased. In addition, the gap between the lower and upper bound values also increased. The results show that for 100 medians the lower bound had increased to 6% of the best known solution and the upper bound value had increased to 11% of the best known solution, with the gap between the upper and lower bounds increasing to 5%.

Crossover Technique 1 was again used on problem set pcb3038 (Reinelt, 1991), consisting of 3,038 nodes expressed as two dimensional cartisian coordinates. The algorithm was run 10 times each for median values 10, 20, 30, 40, 50, 60, 70, 80, and 100. As with problem set fl1400, for each median value a lower bound value was identified representing the best fitness value and an upper bound value was identified representing the worst fitness value for the given median value. Again, an average fitness value was calculated for each median value and a gap value was calculated that represented the percentage deviation between the lower bound value and the upper bound value. Finally an Error Rate was calculated for both the lower bound and upper bound values that represented the deviation of the value from the best known solution. Table 6. **Summary of Results Using pcb3038 Problem Set and Crossover Technique 1** shows an aggregation of the runs and the calculated values.

Table 6. Summary of Results Using pcb3038 Problem Set and Crossover Technique 1

n	p	Best Known	LBOUND	ERR	UBOUND	ERR	Gap	Average	ERR
3038	10	1,213,082.03	1,235,657.95	0.02	1,260,371.11	0.04	0.02	1,247,556.71	0.03
3038	20	840,844.53	866,207.93	0.03	881,377.98	0.05	0.02	881,377.98	0.05
3038	30	677,436.66	701,283.38	0.04	719,511.81	0.06	0.03	707,884.82	0.04
3038	40	571,887.75	595,626.06	0.04	607,961.80	0.06	0.02	602,060.00	0.05
3038	50	507,655.19	529,623.54	0.04	539,099.21	0.06	0.02	535,450.39	0.05
3038	60	460,797.55	484,576.92	0.05	494,878.73	0.07	0.02	490,156.53	0.06
3038	70	426,153.31	448,061.43	0.05	457,397.02	0.07	0.02	452,880.97	0.06
3038	80	397,585.89	419,612.42	0.06	430,868.60	0.08	0.03	424,599.45	0.07
3038	90	373,488.82	396,657.80	0.06	406,429.04	0.09	0.03	401,313.33	0.07
3038	100	352,755.13	380,153.39	0.08	387,810.62	0.10	0.02	384,189.62	0.09

The results show that for 10 medians the lower bound solution was within 2% of the best known solution and the upper bound solution within 4% of the best known solution. The average of all runs for 10 medians was also within 3% of the best known solution and the gap between the upper and lower bounds was no more than 2%. As with problem set fl1400, as the number of medians increases from 10 to 100 the deviation from the best known solution for the upper and lower bounds increased. In addition, the gap between the lower and upper bound values also increased. The results show that for 100 medians the lower bound had increased to 8% of the best known solution and the upper bound value had increased to 10% of the best known solution. As opposed to the results from problem set fl1400, the gap between the lower and upper bounds remained consistent at 2 or 3 percent as the number of medians increased.

In order to compare the algorithm against a larger problem set with best known values published in the literature, Crossover Technique 1 was used on problem set rl5934 (Reinelt, 1991) consisting of 5,934 nodes expressed as two dimensional cartisian coordinates. This problem set has not been extensively used in prior studies, however

Hansen, P., & Mladenovic (2001) did use it for the p-median problem and published the results of their study. The algorithm was run 10 times each for the same set of median values as were used for fl1400 and pcb3038. As with the other problem sets, for each median value a lower bound value was identified representing the best fitness value and an upper bound value was identified representing the worst fitness value for the given median value. Again, an average fitness value was calculated for each median value and a gap value was calculated that represented the percentage deviation between the lower bound value and the upper bound value. An Error Rate was calculated for both the lower bound and upper bound values that represented the deviation of the value from the best known solution. Table 7. **Summary of Results Using rl5934 Problem Set and Crossover Technique 1** shows an aggregation of the runs and the calculated values.

Table 7. Summary of Results Using rl5934 Problem Set and Crossover Technique 1

n	p	Best Known	LBOUND	ERR	UBOUND	ERR	Gap	Average	ERR
5934	10	9,794,951.00	9,948,378.50	0.02	10,147,346.07	0.04	0.02	10,071,242.64	0.03
5934	20	6,729,282.50	6,931,397.86	0.03	7,056,057.80	0.05	0.02	7,001,087.57	0.04
5934	30	5,405,661.50	5,621,758.91	0.04	5,749,487.43	0.06	0.02	5,686,562.44	0.05
5934	40	4,574,374.00	4,788,835.20	0.05	4,861,491.78	0.06	0.02	4,828,020.98	0.06
5934	50	4,053,917.75	4,227,396.73	0.04	4,308,526.23	0.06	0.02	4,269,273.97	0.05
5934	60	3,655,898.75	3,843,454.42	0.05	3,924,787.17	0.07	0.02	3,875,696.53	0.06
5934	70	3,353,885.00	3,538,947.96	0.06	3,612,279.31	0.08	0.02	3,574,529.44	0.07
5934	80	3,104,877.75	3,282,953.12	0.06	3,367,070.46	0.08	0.03	3,336,407.94	0.07
5934	90	2,903,895.25	3,090,483.72	0.06	3,153,436.64	0.09	0.02	3,124,103.23	0.08
5934	100	2,733,817.25	2,925,863.34	0.07	3,000,827.73	0.10	0.03	2,952,281.54	0.08

The results from problem set rl5934 were very similar to the results of rl3038. For 10 medians the lower bound solution was within 2% of the best known solution and the upper bound solution within 4%. The average of all runs for 10 medians was also within

3% of the best known solution and the gap between the upper and lower bounds was no more than 2%. As with the other problem sets, the lower and upper bound error rates increased as the number of medians increased. The gap between the lower and upper bounds remained consistent at between 2 and 3 percent. The average error rate tended slightly toward the upper bound rather than the lower bound. For 100 medians the algorithm performed slightly better for problem set rl5934 than it did for problem set pcb3038.

Summary of Results Using Crossover Technique 2

A second techinque for the crossover operator was also tested. This operator was similar to crossover technique 1 in that it too used location information to swap genes within a local proximity to each other, however, it used a rougher approximation and was less computationally intensive. Crossover techinque 2 is illustrated in Figure 4.

Crossover Technique 2. The same test plan and problem sets were used for crossover technique 2 as were used for crossover techique 1. For each median value, the run that produced the best fitness value was identified as the lower bound. The result that produced the worst fitness function was selected and identified as the Upper Bound. The average of all runs for each median value was also calculated and identified as the average for the repective median value. A Gap value was calculated representing the percentage deviation between the lower bound and the upper bound values. An Error Rate was calculated for both the lower bound and upper bound values that represents the deviation of the value from the best known solution. For each problem set the results were aggreated and are shown in Table 8. **Summary of Results Using fl1400 Problem**

Set and Crossover Technique 2, Table 9. **Summary of Results Using pcb3038 Problem Set and Crossover Technique 2**, and Table 10. **Summary of Results Using rl5934 Problem Set and Crossover Technique 2**.

Table 8. Summary of Results Using fl1400 Problem Set and Crossover Technique 2

n	p	Best Known	LBOUND	ERR	UBOUND	ERR	Gap	Average	ERR
1400	10	101,248.13	103,260.86	0.02	106,808.55	0.05	0.04	105,271.08	0.04
1400	20	57,856.32	59,620.89	0.03	62,147.94	0.07	0.04	60,873.34	0.05
1400	30	44,013.02	46,408.24	0.05	49,152.84	0.12	0.06	47,825.60	0.09
1400	40	35,002.02	37,382.30	0.07	38,803.48	0.11	0.04	38,017.36	0.09
1400	50	29,089.71	31,233.34	0.07	32,246.68	0.11	0.03	31,728.49	0.09
1400	60	25,160.40	26,923.56	0.07	28,724.73	0.14	0.07	27,927.86	0.11
1400	70	22,125.46	23,877.50	0.08	25,255.08	0.14	0.06	24,522.56	0.11
1400	80	19,870.28	21,345.60	0.07	22,544.75	0.13	0.06	22,041.43	0.11
1400	90	17,987.91	19,378.26	0.08	20,575.70	0.14	0.07	20,025.61	0.11
1400	100	16,551.20	18,000.88	0.09	19,086.31	0.15	0.07	18,595.79	0.12

Table 9. Summary of Results Using pcb3038 Problem Set and Crossover Technique 2

n	p	Best Known	LBOUND	ERR	UBOUND	ERR	Gap	Average	ERR
3038	10	1,213,082.03	1,260,629.99	0.04	1,297,922.89	0.07	0.03	1,280,633.14	0.06
3038	20	840,844.53	890,561.86	0.06	916,036.39	0.09	0.03	898,466.94	0.07
3038	30	677,436.66	718,664.68	0.06	739,512.88	0.09	0.03	728,323.61	0.08
3038	40	571,887.75	610,078.42	0.07	627,555.51	0.10	0.03	620,651.24	0.09
3038	50	507,655.19	543,697.60	0.07	559,375.54	0.10	0.03	552,289.50	0.09
3038	60	460,797.55	495,332.18	0.07	511,506.81	0.11	0.04	503,546.17	0.09
3038	70	426,153.31	462,914.87	0.09	471,806.97	0.11	0.02	466,569.02	0.09
3038	80	397,585.89	432,562.96	0.09	445,129.31	0.12	0.03	437,213.79	0.10
3038	90	373,488.82	407,519.87	0.09	420,172.50	0.12	0.03	414,208.96	0.11
3038	100	352,755.13	388,203.24	0.10	402,003.86	0.14	0.04	395,856.02	0.12

Table 10. Summary of Results Using rl5934 Problem Set and Crossover Technique 2

n	p	Best Known	LBOUND	ERR	UBOUND	ERR	Gap	Average	ERR
5934	10	9,794,951.00	10,209,378.78	0.04	10,466,550.99	0.07	0.03	10,348,727.14	0.06
5934	20	6,729,282.50	7,078,046.08	0.05	7,336,289.91	0.09	0.04	7,215,296.71	0.07
5934	30	5,405,661.50	5,810,175.13	0.07	5,921,169.79	0.10	0.02	5,873,645.67	0.09
5934	40	4,574,374.00	4,939,578.00	0.08	5,022,422.26	0.10	0.02	4,976,067.95	0.09
5934	50	4,053,917.75	4,341,185.98	0.07	4,473,520.48	0.10	0.03	4,397,660.06	0.08
5934	60	3,655,898.75	3,937,454.67	0.08	4,055,578.38	0.11	0.03	3,988,343.12	0.09
5934	70	3,353,885.00	3,637,122.42	0.08	3,733,648.25	0.11	0.03	3,681,146.20	0.10
5934	80	3,104,877.75	3,366,446.91	0.08	3,469,206.70	0.12	0.03	3,433,097.16	0.11
5934	90	2,903,895.25	3,167,696.17	0.09	3,267,340.80	0.13	0.03	3,218,160.52	0.11
5934	100	2,733,817.25	2,987,061.70	0.09	3,083,976.22	0.13	0.04	3,038,786.89	0.11

Crossover technique 2 did not perform as well as crossover techinque 1. In general, the results from technique 2 were two to three percent worse than techinque 1. Interestingly, the pattern of the results from both techniques were very similar. For problem set fl1400 using 10 medians the lower bound error rate was two percent above the best known solution and the upper bound error rate was five percent above the best known solution. The error rates increased as the number of medians increased with 100 medians generating a nine percent error rate for the lower bound and a fifteen percent error rate for the upper bound. As with crossover technique 1, the gap in error rates increased steadily as the medians increased from four percent for 10 medians to seven percent for 100 medians. For problem sets pcb3038 and rl5934 the results for crossover technique 2 were inferior to crossover technique 1, however the pattern of the results were very similar. For both problem sets the lower bound results for 10 medians was seven percent off the best known solution and the upper bound results were seven percent off the best known solution. As the medians increased the algorithm performed slightly better for problem set rl5934 than pcb3038. Using 100 medians problem set rl5934 had a

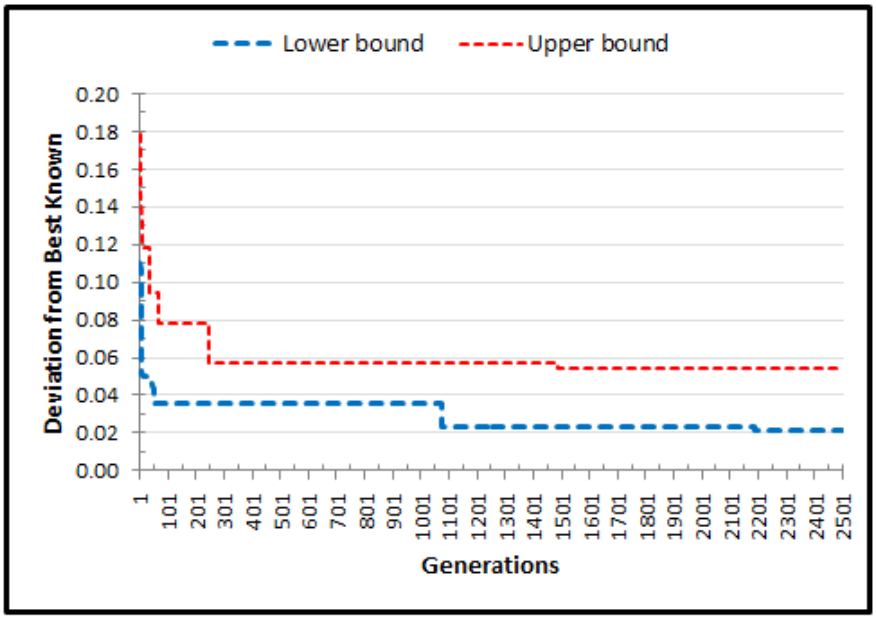
nine percent lower bound error rate and pcb3038 had a ten percent lower bound error rate. The upper bound error rate for rl5934 was thirteen percent and the upper bound error rate for pcb3038 was fourteen percent. Similar to crossover technique 1 the gap in error rates remained consistent as the medians increased for both problem sets ranging from two to four percent. The average error rate for all three problem sets were also very similar with none of them deviating more than twelve percent from the best known solutions. This was still inferior to crossover technique 1 which had average error rates that deviated at most ten percent from the best known solution.

Analysis of Run Profiles Using Crossover Technique 1

The profiles of the DAGA runs were examined in order to gain a better understanding of how efficiently the algorithm evolved from its initial state to an optimal solution. Because the algorithm uses an elitist strategy where the fittest chromosome in each generation is passed on to the succeeding generation, the solution was not expected to degrade at any point in the run. Hypothetically, if the algorithm moved toward the optimal at a constant rate it would exhibit a linear descent. In practice, the solution improves in an uneven stepped fashion. Accelerated improvement results in steeper steps and decelerated improvement results in elongated steps. On the run profile a steeper curve indicates a quicker, in terms of the number of generations, improvement from its initial solution toward an optimal solution. The rate of improvement can be considered an indicator as to the efficiency of the algorithm in searching the problem space and identifying good solutions.

The results produced from the runs against the fl1400 problem set described above were further analyzed and the line graphs were created to illustrate the analysis. For each value of *p* in the test set a line graph was created that tracked two test runs representing the run that produced the lower bound value and the run that produced the upper bound value. In each graph the x-axis represents generations and the y-axis represents the deviation of the fitness value for the given generation, expressed as an error rate, from the best known solution. Each graph was constrained to the first 2500 generations to provide a common basis of comparison between the lower and upper band values as well as the different values of *p*. By converting raw fitness scores into error rates, a consistent basis for comparison is provided across all test instances. This allows some determination to be made about how variations in the algorithm impact its ability to efficiently move to an optimal or near-optimal solution. As part of the graph a table was added that shows the generation count and fitness value each time the fitness value changes. These are essentially the step points in the graph and provide a more complete profile of the run. These values are not constrained to the first 2500 generations but instead are listed until the best value for the run is found.

The run profile for 10 medians is shown in Figure 6. **Run Profile for Problem Set fl1400 with 10 medians**. In this profile the lower bound run starts with an error rate of 11% and the upper bound run starts at 17.8%. Within 100 generations the lower bound run had improved to an error rate of 3.5% and the upper bound run had improved to an error rate of 7.8%. After that point the evolution of the solution slowed significantly, only improving to 2.1% and 5.4% respectively after 2,500 generations.

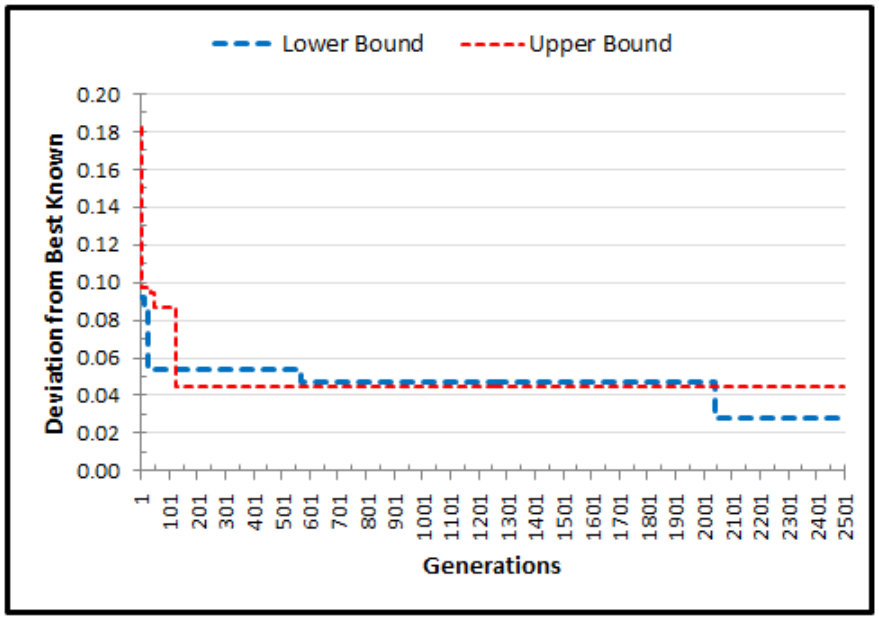


Lower Bound		Upper Bound	
Gen	FitVal	Gen	FitVal
1	112,412.12	1	119,329.66
4	111,980.37	4	116,026.19
7	106,451.62	7	114,362.59
15	106,285.68	11	113,183.64
43	105,843.16	38	110,824.45
49	105,612.05	66	109,248.92
51	104,862.76	69	109,126.04
1077	103,648.44	249	107,002.65
2191	103,399.72	1487	106,719.21
2982	103,359.62		
3094	102,435.59		
3855	102,421.02		
5945	101,253.83		

Figure 6. Run Profile for Problem Set fl1400 with 10 medians

The run profile for 20 medians is shown in Figure 7. **Run Profile for Problem Set fl1400 with 20 medians**. This profile is similar to the runs with 10 medians. The lower bound run starts with an error rate of 9% and the upper bound run starts at 18.3%. Within 100 generations the lower bound run had improved to an error rate of 5.4% and the upper bound run had improved to an error rate of 8.7%. After 2,500 generations the

runs had only improved to error rates of 2.8% and 4.4% respectively. Interestingly, the upper bound run took a big step at 125 generations and was producing a better solution than the lower bound run for a while but then failed to improve any more.

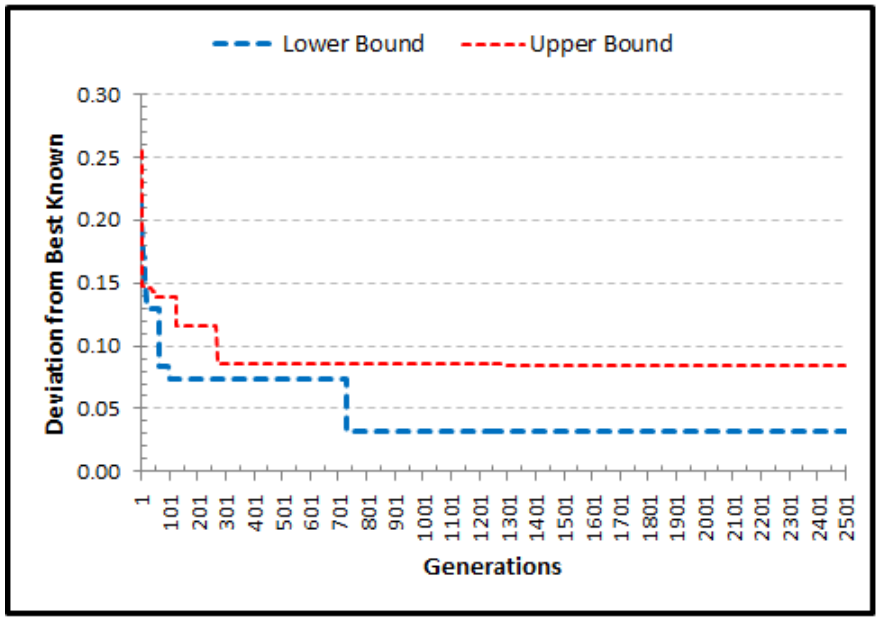


Lower Bound		Upper Bound	
Gen	FitVal	Gen	FitVal
1	63,194.03	1	68,420.12
12	62,798.86	2	63,473.04
24	60,993.44	38	63,293.41
573	60,585.90	50	62,898.79
2044	59,498.15	125	60,449.35
2505	59,295.16		
3042	59,211.94		
3684	58,859.55		

Figure 7. Run Profile for Problem Set fl1400 with 20 medians

For 30 medians the run profiles show a pattern very similar to the prior two run profiles with most of the improvement coming in the first 100 generations. Figure 8. **Run Profile for Problem Set fl1400 with 30 medians** illustrates the run profiles. The lower bound run starts with an error rate of 21.2% and the upper bound run starts at 25.4%.

Within 100 generations the lower bound run had improved to an error rate of 7.3% and the upper bound run had improved to an error rate of 13.8%. After 2,500 generations the runs had further improved to error rates of 3.1% and 8.4% respectively.

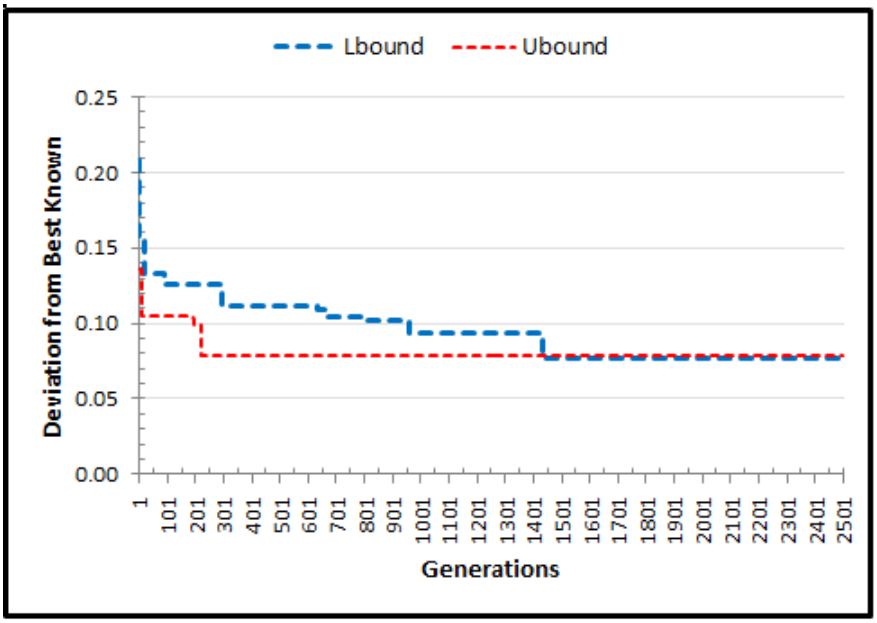


Lower Bound		Upper Bound	
Gen	FitVal	Gen	FitVal
1	53,350.54	1	55,215.57
3	52,814.07	2	51,280.50
4	51,649.83	4	50,528.65
7	51,589.14	21	50,413.09
11	51,454.67	39	50,342.85
14	50,799.90	48	50,120.45
17	49,948.89	127	49,082.53
22	49,849.17	270	47,801.11
23	49,720.25	1123	47,769.18
66	47,667.34	1281	47,729.94
100	47,246.43		
730	45,404.13		

Figure 8. Run Profile for Problem Set fl1400 with 30 medians

The run profiles for 40 medians is shown in Figure 9. **Run Profile for Problem Set fl1400 with 40 medians**. The error rates start at 20.8% and 13.6% for the lower and

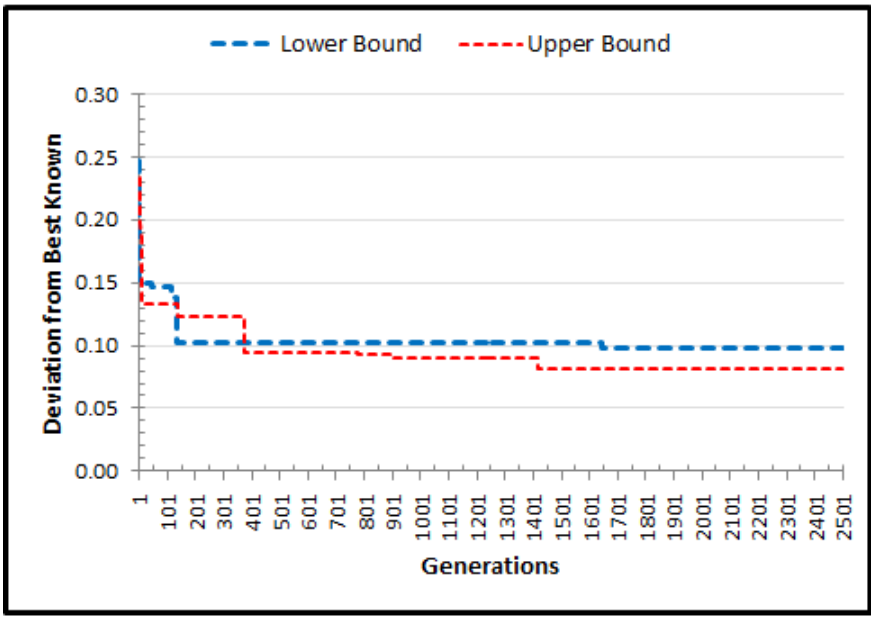
upper bound runs. Within 100 generations the error rates had improved to 12.6% and 10.4%. Interestingly, the upper bound run produced a better value until generation 1,436 when the lower bound run passed it. At 2,500 generations the lower bound run showed a slightly better result at 7.7% versus 7.8%. In this profile the upper bound run did not show comparable efficiency.



Lower Bound		Upper Bound	
Gen	FitVal	Gen	FitVal
1	42,288.72	1	39,776.44
2	40,404.06	10	38,665.58
19	39,668.81	193	38,588.13
88	39,414.57	197	38,450.37
296	38,892.94	221	37,741.65
634	38,813.12		
667	38,637.25		
807	38,576.39		
961	38,291.63		
1436	37,698.92		
2863	37,332.60		
2866	36,514.57		

Figure 9. Run Profile for Problem Set fl1400 with 40 medians

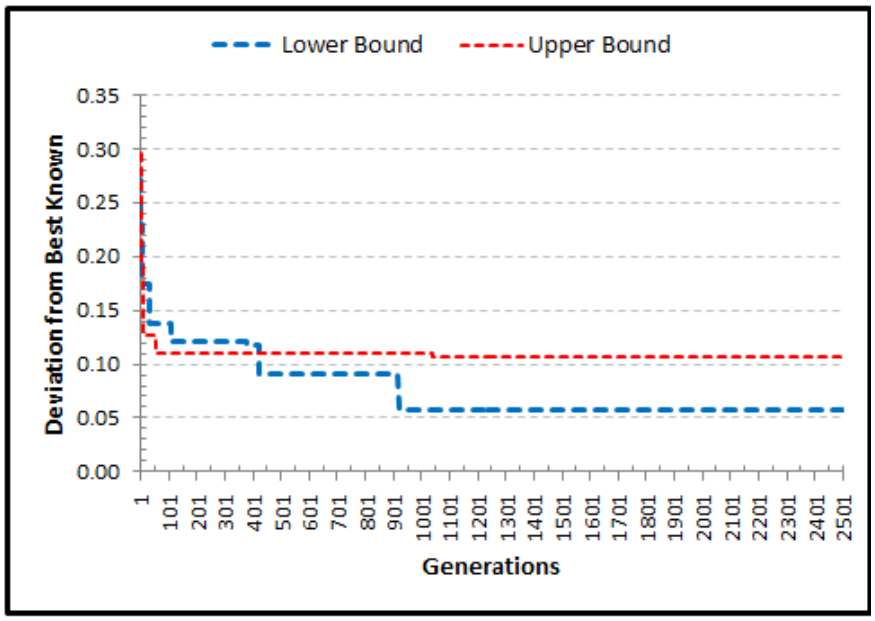
The run profile for 50 medians is shown in Figure 10. **Run Profile for Problem Set fl1400 with 50 medians**. Again, most of the improvements came early in the run. For the lower bound, 49.7% of the gains came in the first 100 generations and 72% came in the first 500. For the upper bound, 65.8% of the gains came in the first 100 generations and 95.5% came in the first 500.



Lower Bound		Upper Bound	
Gen	FitVal	Gen	FitVal
1	36,266.54	1	35,871.58
2	33,419.60	2	35,701.08
44	33,370.14	6	34,847.76
115	33,115.32	7	34,443.19
136	32,068.41	8	32,973.59
1647	31,934.10	139	32,662.59
4076	31,689.57	375	31,831.36
5813	30,705.57	777	31,785.99
7825	30,435.82	893	31,696.72
		1416	31,464.58

Figure 10. Run Profile for Problem Set fl1400 with 50 medians

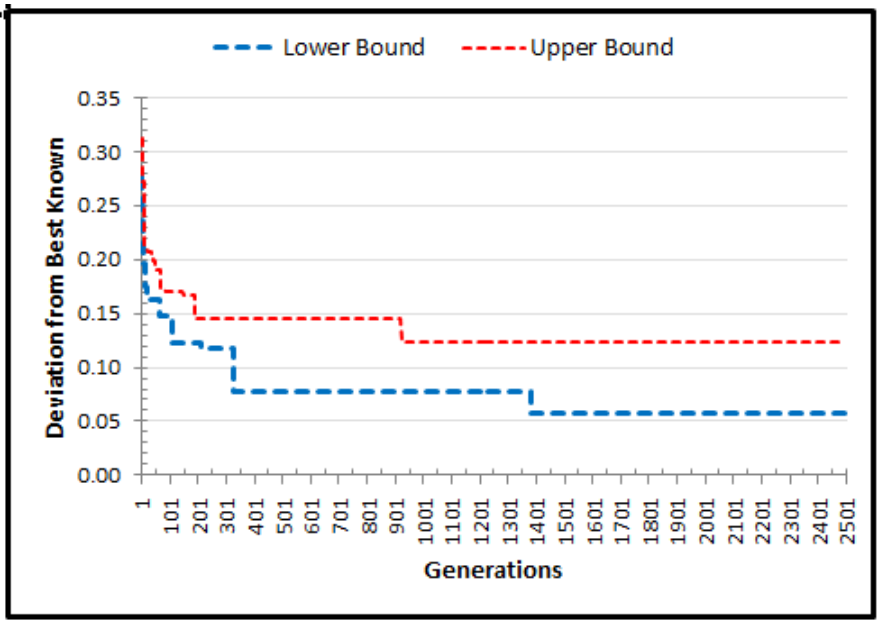
The 60 median run is shown in Figure 11. **Run Profile for Problem Set fl1400 with 60 medians**. As with the eariler runs the algorithm shows good efficiency early and than slows rapidly. For the lower bound, 62.8% of the gains came in the first 100 generations and 84.9% came in the first 500. For the upper bound, 95.2% of the gains came in the first 100 generations and further gains did not occurr until generation 1039.



Lower Bound		Upper Bound	
Gen	FitVal	Gen	FitVal
1	31,998.31	1	32,582.66
3	30,950.21	2	32,072.12
5	30,387.54	3	31,100.11
7	29,566.26	6	30,023.34
33	28,620.62	11	28,358.98
111	28,199.55	56	28,182.70
381	28,140.07	57	27,917.60
424	27,433.28	1039	27,838.88
919	26,620.11	2559	27,682.85

Figure 11. Run Profile for Problem Set fl1400 with 60 medians

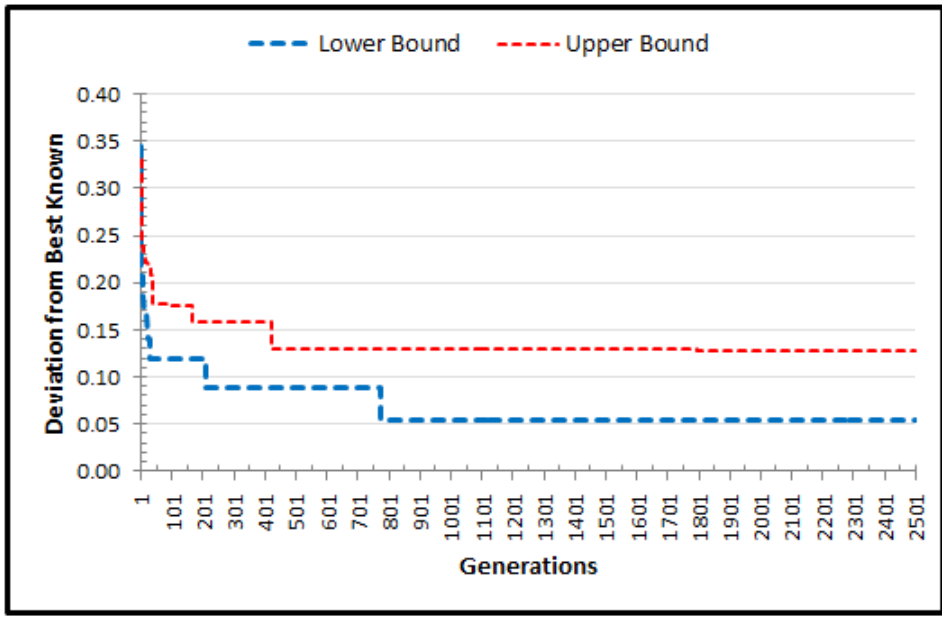
The 70 median run is shown in Figure 12. **Run Profile for Problem Set fl1400 with 70 medians**. It shows good efficiency early and then again slows rapidly after 200 or 300 generations. For the lower bound, 62.8% of the gains came in the first 100 generations and 84.9% came in the first 500. For the upper bound, 95.2% of the gains came in the first 100 generations and further gains did not occurr until generation 1039.



Lower Bound		Upper Bound	
Gen	FitVal	Gen	FitVal
1	28,235.12	1	29,036.03
3	28,140.90	3	28,582.62
4	26,908.17	6	28,152.76
5	26,476.83	9	27,492.51
15	25,992.45	10	27,460.04
18	25,719.45	11	26,825.56
66	25,398.36	20	26,708.10
109	24,839.12	37	26,550.04
213	24,746.70	40	26,516.22
326	23,836.25	48	26,322.69
1384	23,412.64	67	25,941.00
		84	25,888.93
		148	25,835.75
		188	25,349.43
		922	24,869.24

Figure 12. Run Profile for Problem Set fl1400 with 70 medians

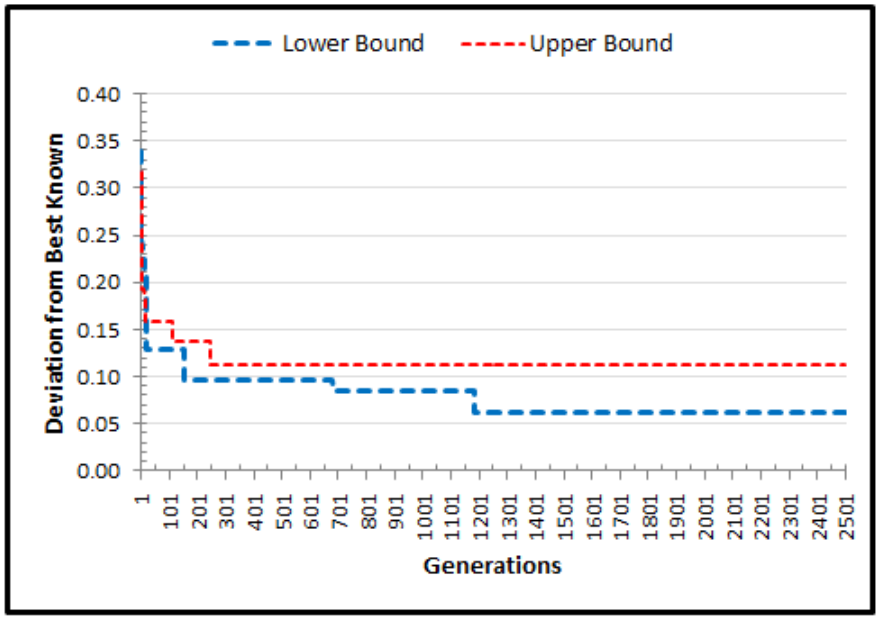
The run profile for 80 medians is shown in Figure 13. **Run Profile for Problem Set fl1400 with 80 medians** and shows a similar pattern as the other runs.



Lower Bound		Upper Bound	
Gen	FitVal	Gen	FitVal
1	26,702.36	1	26,434.02
2	25,580.31	2	25,418.36
3	23,978.62	3	24,948.07
9	23,566.91	4	24,712.85
10	23,198.30	5	24,600.88
21	22,666.42	11	24,478.65
28	22,242.63	13	24,363.84
209	21,942.89	15	24,272.72
212	21,626.11	24	24,195.97
776	20,958.67	33	24,025.52
		34	23,985.48
		39	23,398.83
		70	23,378.10
		91	23,355.75
		165	23,012.30
		425	22,450.87
		1794	22,421.96
		2982	22,321.11
		4452	22,280.45

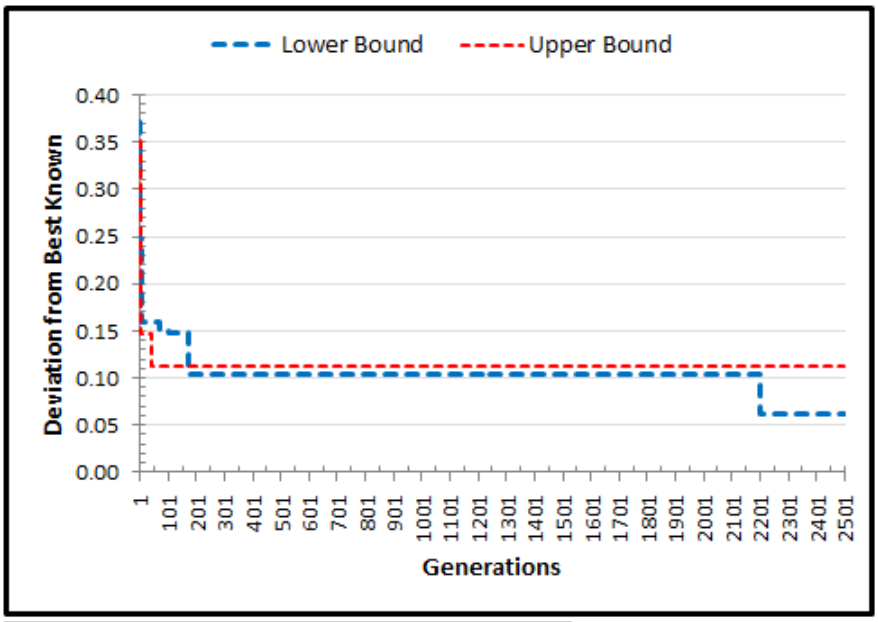
Figure 13. Run Profile for Problem Set fl1400 with 80 medians

The run profiles for 90 and 100 medians against the fl1400 problem set are shown in Figure 14. **Run Profile for Problem Set fl1400 with 90 medians** and Figure 15. **Run Profile for Problem Set fl1400 with 100 medians**, respectively. Even more than the other runs, these two runs show great efficiency early with over 70% of the improvement coming in the first 100 generations. After that, the progress slows markedly especially for the upper bound runs.



Lower Bound		Upper Bound	
Gen	FitVal	Gen	FitVal
1	24,087.98	1	23,701.46
2	22,410.75	4	21,465.65
4	22,229.17	14	21,287.36
7	22,019.03	19	20,824.18
13	21,829.85	113	20,446.90
19	20,881.40	245	20,025.31
20	20,309.57		
155	19,711.91		
682	19,524.13		
1182	19,085.52		

Figure 14. Run Profile for Problem Set fl1400 with 90 medians



Lower Bound		Upper Bound	
Gen	FitVal	Gen	FitVal
1	22,694.86	1	22,378.51
2	21,776.22	3	22,084.34
3	20,649.70	4	20,624.10
7	19,186.55	5	19,963.96
72	19,048.75	6	18,985.25
104	19,001.81	44	18,423.08
171	18,985.81		
172	18,261.34		
2204	17,580.43		

Figure 15. Run Profile for Problem Set fl1400 with 100 medians

All of the runs showed roughly the same pattern. Most of the progress, at least 50% in every case, is made in the first 100 generations. After that the progress started to slow and after 500 generations at least 70% of the progress had been made for every run. After 500 generations progress was very slow, if at all, with many generations necessary

to find the next step. This pattern of early efficiency and then rapid decline seems to be an indicator that the algorithm is consistently getting trapped in a local optimum.

Summary of Results Using an Unstructured Initial Generation

An additional test was created to determine the impact of the technique used by DAGA to create the initial generation on the overall efficacy of the algorithm. The initial generation creation technique used by DAGA partitioned the problem set into spatially-oriented regions and selected nodes from each region evenly to create the chromosomes that populated the initial generation. Refer to the Algorithm Design section in the Methodology Chapter of this paper for a more detailed description of the technique used by DAGA. This technique created a structured initial generation. In order to test the efficacy of this technique, a new algorithm was created that creates the initial generation by randomly selecting nodes from the problem set and building chromosomes until the initial generation was fully populated. This is the technique used in the canonical and most other genetic algorithms used for the *p*-median problem. With the exception of the technique used for the initial generation, all other aspects of the algorithm were identical to DAGA using Crossover Operator Technique 1. This new algorithm was identified as DAGA-IG. The modified algorithm was run ten times each for *p* values 10 through 100 in the fl1400 problem set. The lower and upper bound results were compared with the lower and upper bound results from DAGA using crossover technique 1 and the Best Known results from literature. In addition, the results were graphed to compare the efficiency of the modified algorithms as compared to DAGA. Table 11. **Fitness Values Using an Unstructured Initial Generation** and Table 12. **Error Rates Using an**

Unstructured Initial Generation compare and summarize the results generated by DAGA-IG with the results produced by DAGA and the best known results. Table 13. **Deviation From DAGA When Using an Unstructured Initial Generation** compares the results produced by DAGA-IG directly with the results produced by DAGA.

Table 11. Fitness Values Using an Unstructured Initial Generation

		DAGA			Random Initial Generation		
p	Best Known	Lower Bound	Upper Bound	Average	Lower Bound	Upper Bound	Average
10	101,248.13	101,248.57	102,711.90	102,148.43	101,804.66	103,745.91	102,911.92
20	57,856.32	58,859.55	60,449.35	59,600.23	58,543.26	60,491.51	59,873.35
30	44,013.02	45,404.13	47,729.94	46,477.33	44,311.24	46,838.95	46,209.36
40	35,002.02	36,514.57	37,741.65	37,094.65	36,453.38	38,408.52	37,206.61
50	29,089.71	30,240.72	31,262.88	30,883.38	31,080.68	31,893.96	31,569.41
60	25,160.40	26,620.11	27,682.85	27,204.38	26,588.45	27,606.66	27,001.97
70	22,125.46	23,412.64	24,869.24	24,034.39	23,578.34	24,465.27	24,026.07
80	19,870.28	20,958.67	22,280.45	21,664.30	21,571.70	22,523.63	21,956.95
90	17,987.91	19,085.52	20,025.31	19,528.80	18,736.86	20,597.33	19,735.83
100	16,551.20	17,580.43	18,423.08	18,129.04	17,850.18	18,912.10	18,256.67

Table 12. Error Rates Using an Unstructured Initial Generation

	DAGA			Random Initial Generation		
p	Lower Bound	Upper Bound	Avg	Lower Bound	Upper Bound	Avg
10	0.0000	0.0145	0.0089	0.0055	0.0247	0.0164
20	0.0173	0.0448	0.0301	0.0119	0.0455	0.0349
30	0.0316	0.0845	0.0560	0.0068	0.0642	0.0499
40	0.0432	0.0783	0.0598	0.0415	0.0973	0.0630
50	0.0396	0.0747	0.0617	0.0684	0.0964	0.0852
60	0.0580	0.1003	0.0812	0.0568	0.0972	0.0732
70	0.0582	0.1240	0.0863	0.0657	0.1058	0.0859
80	0.0548	0.1213	0.0903	0.0856	0.1335	0.1050
90	0.0610	0.1133	0.0857	0.0416	0.1451	0.0972
100	0.0622	0.1131	0.0953	0.0785	0.1426	0.1030

Table 13. Deviation From DAGA When Using an Unstructured Initial Generation

	Random Initial Generation		
p	Lower Bound	Upper Bound	Avg
10	0.0055	0.0101	0.0075
20	-0.0054	0.0007	0.0046
30	-0.0241	-0.0187	-0.0058
40	-0.0017	0.0177	0.0030
50	0.0278	0.0202	0.0222
60	-0.0012	-0.0028	-0.0074
70	0.0071	-0.0162	-0.0003
80	0.0292	0.0109	0.0135
90	-0.0183	0.0286	0.0106
100	0.0153	0.0265	0.0070

The results produced by DAGA-IG were usually inferior to those produced by DAGA but only slightly inferior. This result was consistent with expectations. A structured initial generation would be expected to provide a better starting solution but not necessarily a better final solution. The structured approach's value to the algorithm is to make the algorithm more efficent by providing a superior starting point. To illustrate this, the run profiles from DAGA-IG are compared with the run profiles from DAGA. In this comparison the lower bound results from both algorithms are tracked on a single graph and the stepped solutions are accumulated into an associated table. All of the run profiles are shown in Appendix A. The results of the comparison show that for almost every median count the starting position of DAGA is significantly superior to DAGA-IG. It also shows that the while the structured approach provides better efficiency early in the run, that by 2,500 generations that advantage is largely gone and the end-state does not consistently vary in a significant way.

Summary of Results Using a Random Crossover Operator

A test was created to determine the impact of the crossover operator used by DAGA to create candidate chromosomes for the next generation on the generated solutions. DAGA tested two similar but distinct crossover operators. The test showed that the first technique consistently produced better results. This crossover operator selected a gene from each of the parent chromosomes that were spatially close to each other in the search space to swap in the candidate chromosomes. Refer to the Algorithm Design section in the Methodology Chapter of this paper for a more detailed description of crossover operator 1 used by DAGA. The operator is illustrated in Figure 3. **Crossover Technique 1**. In order to test the impact of this crossover operator on DAGA a new algorithm was created that used a crossover operator that randomly selected genes for crossover with no bias for their location. This algorithm was designated as DAGA-CO. The DAGA-CO crossover operator is functionally similar to the technique used by most other genetic algorithms in the literature used for the *p*-median problem. With the exception of the crossover operator, all other aspects of the algorithm were identical to DAGA using Crossover Operator Technique 1. The modified algorithm was run ten times each for *p* values 10 through 100 in the fl1400 problem set. The lower and upper bound results were compared with the lower and upper bound results from DAGA using crossover technique 1 and the Best Known results from literature. Table 14. **Fitness Values Using a Random Crossover Operator** and Table 15. **Error Rates Using Random Crossover Operator** compares and summarizes the results generated by DAGA-CO with the results produced by DAGA and the best known results. Table 16.

Deviation From DAGA When Using a Random Crossover Operator compares the results produced by DAGA-CO directly with the results produced by DAGA.

Table 14. Fitness Values Using a Random Crossover Operator

		DAGA			Random Crossover		
p	**Best Known**	**Lower Bound**	**Upper Bound**	**Average**	**Lower Bound**	**Upper Bound**	**Average**
10	101,248.13	101,248.57	102,711.90	102,148.43	103,112.17	107,799.61	105,946.88
20	57,856.32	58,859.55	60,449.35	59,600.23	61,465.61	64,172.08	62,847.82
30	44,013.02	45,404.13	47,729.94	46,477.33	48,497.59	50,646.75	49,701.74
40	35,002.02	36,514.57	37,741.65	37,094.65	38,474.95	40,794.54	39,839.03
50	29,089.71	30,240.72	31,262.88	30,883.38	33,357.79	35,084.50	33,965.63
60	25,160.40	26,620.11	27,682.85	27,204.38	28,377.56	30,206.09	29,722.55
70	22,125.46	23,412.64	24,869.24	24,034.39	25,587.96	26,923.99	26,264.89
80	19,870.28	20,958.67	22,280.45	21,664.30	23,009.31	25,135.21	24,290.48
90	17,987.91	19,085.52	20,025.31	19,528.80	21,271.45	22,535.11	21,904.18
100	16,551.20	17,580.43	18,423.08	18,129.04	19,383.77	20,751.34	20,162.75

Table 15. Error Rates Using Random Crossover Operator

	DAGA			Random Crossover		
p	**Lower Bound**	**Upper Bound**	**Avg**	**Lower Bound**	**Upper Bound**	**Avg**
10	0.0000	0.0145	0.0089	0.0184	0.0647	0.0464
20	0.0173	0.0448	0.0301	0.0624	0.1092	0.0863
30	0.0316	0.0845	0.0560	0.1019	0.1507	0.1293
40	0.0432	0.0783	0.0598	0.0992	0.1655	0.1382
50	0.0396	0.0747	0.0617	0.1467	0.2061	0.1676
60	0.0580	0.1003	0.0812	0.1279	0.2005	0.1813
70	0.0582	0.1240	0.0863	0.1565	0.2169	0.1871
80	0.0548	0.1213	0.0903	0.1580	0.2650	0.2225
90	0.0610	0.1133	0.0857	0.1825	0.2528	0.2177
100	0.0622	0.1131	0.0953	0.1711	0.2538	0.2182

Table 16. Deviation From DAGA When Using a Random Crossover Operator

	Random Crossover		
p	Lower Bound	Upper Bound	Avg
10	0.0184	0.0495	0.0372
20	0.0443	0.0616	0.0545
30	0.0681	0.0611	0.0694
40	0.0537	0.0809	0.0740
50	0.1031	0.1222	0.0998
60	0.0660	0.0911	0.0926
70	0.0929	0.0826	0.0928
80	0.0978	0.1281	0.1212
90	0.1145	0.1253	0.1216
100	0.1026	0.1264	0.1122

The results produced by DAGA-CO were significantly inferior to those produced by DAGA for all of the values of *p* tested. The results support the thesis of this study which was that using characteristics of the problem set, in this case location, could have a positive impact on a genetic algorithm. The results of the comparison show that for every median count tested the results produced by DAGA were at least 75% better than DAGA-CO. Given that the crossover operator was the only difference between DAGA and DAGA-CO it is reasonable to conclude that the crossover operator implemented in DAGA was a significant factor in the results it produced.

Summary of Results Using a Random Mutation Operator

A final test was created to determine the impact of the mutation operator used by DAGA on the overall effectiveness of the algorithm. The mutation operator is used in DAGA to introduce new genes into a subset of candidate chromosomes that were not part

of the related parent chromosomes. The primary purposes of the mutation operator is to encourage a more complete search of the problem set and to discourage the algorithm from becoming focused exclusively on a locally but not globally optimal solution. Refer to the Algorithm Design section in the Methodology Chapter of this paper for a more detailed description of the technique used by DAGA. The mutation operator used by DAGA is illustrated in Figure 5. **Mutation Operator**. To test the impact of this mutation operator on DAGA a new algorithm was created that used a mutation operator that randomly selected candidate chromosomes and genes within those chromosomes for mutation with no bias for prior use or their location. This algorithm was designated as DAGA-MU. The mutation operator used in DAGA-MU is functionally similar to the technique used by most other genetic algorithms in the literature used for the *p*-median problem. With the exception of the mutation operator, all other aspects of the algorithm were identical to DAGA using Crossover Operator Technique 1. The modified algorithm was run ten times each for *p* values 10 through 100 in the fl1400 problem set. The lower and upper bound results were compared with the lower and upper bound results from DAGA using crossover technique 1 and the Best Known results from literature. Table 17. **Fitness Values using a Random Mutation Operator** and

Table 18. **Error Rates Using a Random Mutation Operator** compares and summarizes the results generated by DAGA-MU with the results produced by DAGA and the best known results. Table 19. **Deviation From DAGA When Using a Random Mutation Operator** compares the results produced by DAGA-MU directly with the results produced by DAGA.

Table 17. Fitness Values using a Random Mutation Operator

		DAGA			Random Mutation		
p	Best Known	Lower Bound	Upper Bound	Average	Lower Bound	Upper Bound	Average
10	101,248.13	101,248.57	102,711.90	102,148.43	101,841.67	104,657.19	103,337.21
20	57,856.32	58,859.55	60,449.35	59,600.23	58,461.03	61,747.33	60,416.34
30	44,013.02	45,404.13	47,729.94	46,477.33	45,682.05	46,979.75	46,398.42
40	35,002.02	36,514.57	37,741.65	37,094.65	36,253.23	37,870.62	37,089.90
50	29,089.71	30,240.72	31,262.88	30,883.38	30,492.70	31,953.02	31,402.63
60	25,160.40	26,620.11	27,682.85	27,204.38	26,379.66	27,173.60	26,689.61
70	22,125.46	23,412.64	24,869.24	24,034.39	23,180.84	24,067.28	23,589.17
80	19,870.28	20,958.67	22,280.45	21,664.30	21,289.82	21,953.28	21,660.94
90	17,987.91	19,085.52	20,025.31	19,528.80	19,341.48	19,647.26	19,482.08
100	16,551.20	17,580.43	18,423.08	18,129.04	17,436.57	18,070.48	17,728.47

Table 18. Error Rates Using a Random Mutation Operator

	DAGA			Random Mutation		
p	Lower Bound	Upper Bound	Avg	Lower Bound	Upper Bound	Avg
10	0.0000	0.0145	0.0089	0.0059	0.0337	0.0206
20	0.0173	0.0448	0.0301	0.0105	0.0673	0.0442
30	0.0316	0.0845	0.0560	0.0379	0.0674	0.0542
40	0.0432	0.0783	0.0598	0.0357	0.0820	0.0597
50	0.0396	0.0747	0.0617	0.0482	0.0984	0.0795
60	0.0580	0.1003	0.0812	0.0485	0.0800	0.0608
70	0.0582	0.1240	0.0863	0.0477	0.0878	0.0662
80	0.0548	0.1213	0.0903	0.0714	0.1048	0.0901
90	0.0610	0.1133	0.0857	0.0752	0.0922	0.0831
100	0.0622	0.1131	0.0953	0.0535	0.0918	0.0711

Table 19. Deviation From DAGA When Using a Random Mutation Operator

	Random Mutation		
p	Lower Bound	Upper Bound	Avg
10	0.0059	0.0189	0.0116
20	-0.0068	0.0215	0.0137
30	0.0061	-0.0157	-0.0017
40	-0.0072	0.0034	-0.0001
50	0.0083	0.0221	0.0168
60	-0.0090	-0.0184	-0.0189
70	-0.0099	-0.0322	-0.0185
80	0.0158	-0.0147	-0.0002
90	0.0134	-0.0189	-0.0024
100	-0.0082	-0.0191	-0.0221

The results produced by DAGA-MU did not vary significantly from the results produced by DAGA using crossover operator 1 when run against the fl1400 problem set. Though the differences were not large, DAGA-MU using a random techinque for the mutation operator produced slightly better results than those produced by DAGA for all of the values of *p* tested. Given these results it is reasonable to conclude that a mutation operator that uses domain knowledge, specifically the spatial attributes of the problem set, does not significantly improve the genetic algorithm. In fact, the results seem to support the theory that a completely random mutation operator produces better results than a directed mutation operator. This is not completely unexpected given that the purpose of the mutation operator is to introduce diversity into the algorithm. An algorithm like DAGA aggressively focuses on locally optimal solutions through it's crossover operator and a mutation operator that reinforces that local search would not tend to introduce as much diversity as a random operator.

Chapter 5

Conclusions, Implications, Recommendations, and Summary

Conclusions

The research goal of this study was to examine the impact of integrating domain knowledge into a genetic algorithm as applied to the *p*-median problem. The genetic algorithm that was created, DAGA, uses a method for encoding that incorporates spatial location; creates a structured initial population using domain knowledge; is biased toward fitter chromosomes when selecting mating pairs; generates offspring with a spatially sensitive crossover operator; and ensures diversity with a mutation operator that is both biased and spatially sensitive. Using problem sets that have published "best known" solutions, the study examined solutions produced by DAGA in terms of accuracy, performance characteristics, and the contribution of each of the new operators.

DAGA was able to produce good solutions for a variety of problem sets and medians. In somes cases, specifically for smaller problem sets and smaller median counts and using Crossover Technique 1, the solutions produced were optimal or very near optimal, assuming the best known results in the literature are optimal. In all cases tested the solutions produced were good, with the worst solution produced from any test run not deviating from the optimal solution by more than 15%. DAGA's tendency to produce good solutions is further supported by the fact that for all 300 test runs using Crossover Technique 1, the solutions produced for 93% of them were within 10% of the best known

solution. Stated another way, DAGA has a 93% probability of producing a solution for any *p*-median that is within 10% of the optimal solution.

The first crossover technique, which swaps genes in parent chromosomes based on their proximity to each other in the search space, consistently produced better solutions than the second crossover technique, which swaps genes in parent chromosomes on opposite ends of one axis in the search space. It's reasonable to conclude that the first crossover technique places a higher emphasis on domain knowledge and, as a result, it produces better solutions. This conclusion tends to support the hypothesis that using domain knowledge does improve the algorithm.

The smallest problem set, fl1400, and the smallest median count, 10, produced the best solutions. As the median count increased, the solutions deviation from the best known solution also increased. The deviation from the best known also increased as the problem set got larger. Problem set fl1400 produced better solutions than pcb3038, and pcb3038 produced better solutions than rl5934, but only slightly better. These results are probably caused by an exponential increase in the search space as *p* and *n* increases. Given that the stopping criteria used by DAGA is not a function of *n* or *p*, it is likely that a smaller portion of the search space is evaluated as the search space grows.

DAGA was consistently able to produce solutions within 10% of the best known solution in less than 500 generations. After that, the improvement rate slowed significantly. Some of this can be explained by DAGA's use of a structured initial population that partitions the search space spatially and selects chromosomes for the initial population that are distributed across those partitions. This explains why DAGA starts with a relatively good solution but it doesn't explain why it improves rapidily in the

early generations and then slows its improvement in later generations. This is better explained by its technique for selecting parent chromosomes which is biased toward chromosomes with better fitness values. This is not a technique unique to DAGA. Other genetic algorithms that have incorporated hueristics have used a similar approach for parent selection (Correa, et al., 2001). This bias, however, would tend to focus the search on better solutions and result in an accelerated move toward optimal solutions. Similarly, DAGA's use of an "elitist" technique which passes the fittest chromosome from a parent generation to the next generation would tend to slow improvement as the solution moves closer to an optimal. Again, this elitist technique is not unique to DAGA and has been incorporated into other genetic algorithms that have used hueristics (Estivill-Castro & Torres-Velázquez, 1999).

This study introduced three unique techniques to the genetic algorithm; a domain aware structured initial population, a domain aware crossover operator, and a domain aware mutation operator. As part of the study, each of these technques were isolated and tested to determine their impact on the algorithm. The structured initial population improved the initial efficiency of DAGA but had a minimal impact on the resulting solution. In addition, the results produced by starting with a random initial population were equivilant to the results produced by the structured initial population in relatively few generations. Typically less then 500 generations. If the objective of the algorithm is to produce an optimal or near optimal solution, the structured initial population doesn't provide a significant value. On the other hand, if the objective is to produce a good solution in as few generations as possible, the structured technique does add some value.

By isolating the crossover operator, the study shows that a domain aware crossover operator can provide improved results. The results generated by DAGA were better than the results produced by the algorithm that substitued a random crossover operator for the domain area crossover operator. Based on these results, it's reasonable to conclude that a domain aware crossover operator has significant value when building a genetic algorithm to solve the *p*-median problem.

The domain aware mutation operator was shown to produce slightly inferior results than a random mutation operator. When the domain aware mutation operator was replaced with a random mutation operator the resulting solutions were somewhat improved in a majority of the test cases. The purpose of the mutation operator is to introduce diversity into the search and reduce the probability of the algorithm getting stuck on a local optimum. The domain aware crossover operator aggressively focuses on local search. It appears that when the mutation operator reinforces that local search its value is reduced.

The test results suggest the following conclusions about DAGA. First, DAGA is capable of producing solutions for the *p*-median problem with a high degree of accuracy. Next, DAGA is capable of efficiently exploring the search space and finding a good solution to the *p*-median problem in a relatively few generations. Finally, of the three unique characteristics of DAGA, the domain aware crossover operator has the greatest impact on the outcome of the algorithm.

Implications

This dissertation has shown that incorporating inherent properties of the problem into the design of a genetic algorithm can add value to the algorithm while maintaining its core structure. Genetic algrorithms have been shown to be useful in solving NP-hard problems (Goldberg, 1989) including the *p*-median problem. Prior studies have shown that decisions made by the researchers with regards to features of the algorithm such as encoding and operators have a significant impact on the efficacy of the algorithm (Alp, et al., 2003). This study takes that research a step further and shows that incorporating innate properties of the problem into design can also have a positive impact on the efficacy of the algorithm. The findings in this dissertation may prove useful for further studies on the use of genetic algorithms for solving the *p*-median problem. It may also prove useful in the further study of applying genetic algorithms on NP-hard problems other than the *p*-median problem that have inherent characteristics that can be incorporated into the design of the algorithm.

Recommendations

Although this study has shown that integrating domain knowledge about the *p*-median problem into the design of a genetic algorithm can be effective, it is likely that there is more to discover. The study was limited to three problem sets containing two-dimensional cartesian coordinates. Further research is necessary to determine if DAGA would perform similiarly on a wide range of problem sets including small sets, very large sets, and *n*-dimensional sets.

It may also be useful to compare DAGA directly with other metahueristic approaches to the *p*-median problem. Good surveys have been completed (Mladenovi, et al., 2007; Reese, 2005), however, a study that incorporates the same problem sets, programming methods, and run-time infrastructure could provide useful information about the relative strengths and weaknesses of the respective approaches.

Another area for further research is DAGA's applicability to other NP-hard problems. The p-median problem lended itself well to the domain aware approach because of its inherent spatial characteristics. Other NP-hard problems have those same characteristics, such as the Traveling Salesmen Problem, or the K-means problem. Beyond these spatially oriented problems there may be other NP-hard problems with inherent characteristics that can be incorporated into an algorithm design.

There are also areas of further research within the DAGA algorithm. Two crossover techniques were tested, but there are certainly other crossover techniques that take advantage of the domain knowledge that could also be researched. The domain aware mutation operator used in DAGA was not effective in improving the solutions generated. Perhaps further research on mutation operators in genetic algorithms would yield an operator that used domain knowledge to encourage diversity in the search.

The recommended research in this chapter is undoubtably an incomplete list. Optimization problems and genetic algorithms are interesting problems that lend themselves to extensive research. This study represents just one variation of this research. It is the hope of this researcher that it can be used to inspire even more variations.

Summary

The objective of this dissertation was to examine the impact of integrating domain knowledge into a genetic algorithm as applied to the *p*-median problem. To do this, a new genetic algorithm was developed and referred to as DAGA. DAGA differed from the canonical genetic algorithm in a few key ways. Those differentiators are:

1. A technique for encoding the problem set that incorporated the spatial characteristics of the problem members.
2. A structured initial population created by spatially partitioning the search space and creating the initial candidate solutions from that partitioned space.
3. A selection operator that is biased toward fitter solutions when selecting solutions for crossover processing.
4. A crossover operator that considers the location of the problem members when deciding which members to swap in the crossover operation.
5. A mutation operator that is biased toward problem members that are underrepresented in candidate solutions and that considers the location of the members when deciding which to subject to mutation.

Of these five distinguishing characteristics of DAGA, three incorporated domain knowledge about the p-median problem that can be said to be unique to this dissertation at the time of its publication. Those unique characteristics are: A structured initial population based on a spatially partitioned search space; A crossover operator that incorporated location into its decision making process; A mutation operator that incorporated location into its decision making process.

A test plan to examine the impact of these unique elements was developed and DAGA was applied over 900 times. Using published problem sets that have established "best known" solutions for the *p*-median problem, DAGA was applied to several instances of these problem sets using median counts ranging from 10 to 100. The results of the testing showed that DAGA was able to consistently produce accurate solutions. Smaller problem instances with low median counts produced the best results but even worst case results were within 15% of the best known solution and over 90% of the solutions produced were within 10% of the best known solution. DAGA was also able to produce good if not optimal solutions efficiently. In the majority of the test runs, DAGA was able to produce a solution within 10% of optimal in less than 500 generations. After 500 generations the evolution of the optimal solution did slow considerably, with some test runs taking over 10,000 generations before they satisfied the stopping criterion.

Two different crossover operators were tested. The first, identified as Crossover Technique 1, swapped individual members in solutions selected for crossover based on their proximity to each other in the search space. The second, identified as Crossover Technique 2, swapped sets of problem members based on where they were located along a single axis of the search space. Crossover Technique 1 was much more computationally intense and consistently produced more accurate solutions. Crossover Technique 2 did not require as many computational resources as Crossover Technique 1 but it consistently produced inferior results. As a result of these test, further testing of the algorithm was limited to using Crossover Technique 1.

The three key components of DAGA were tested individually to gauge their impact on the overall algorithm. Three new algorithms were created using DAGA as a

basis. The first algorithm substituted a random technique for creating the initial population. The next algorithm replaced the crossover operator with one that randomly selected members from candidate solutions for crossover. The last algorithm randomly selected solutions and members for mutation. These algorithms were each run against the fl1400 problem set using median counts from 10 to 100. The results of these runs were then compared with the results generated by DAGA in earlier tests. The solutions generated by the algorithm using a random approach for the initial population were similar to the solutions produced by DAGA. However, the run profiles showed that DAGA started with a superior solution and performed better in the early generations. This advantage was typically minimized within 500 generations and from there the algorithms performed similarly. The solutions generated by the algorithm using a random technique for crossover were significantly inferior to the solutions produced by DAGA. The solutions generated by the algorithm that used a random technique for mutation sometimes produced solutions that were better than the solutions produced by DAGA. The differences were not generally large and were not consistent but they were enough to suggest that a random approach to mutation is superior to the domain aware technique used by DAGA.

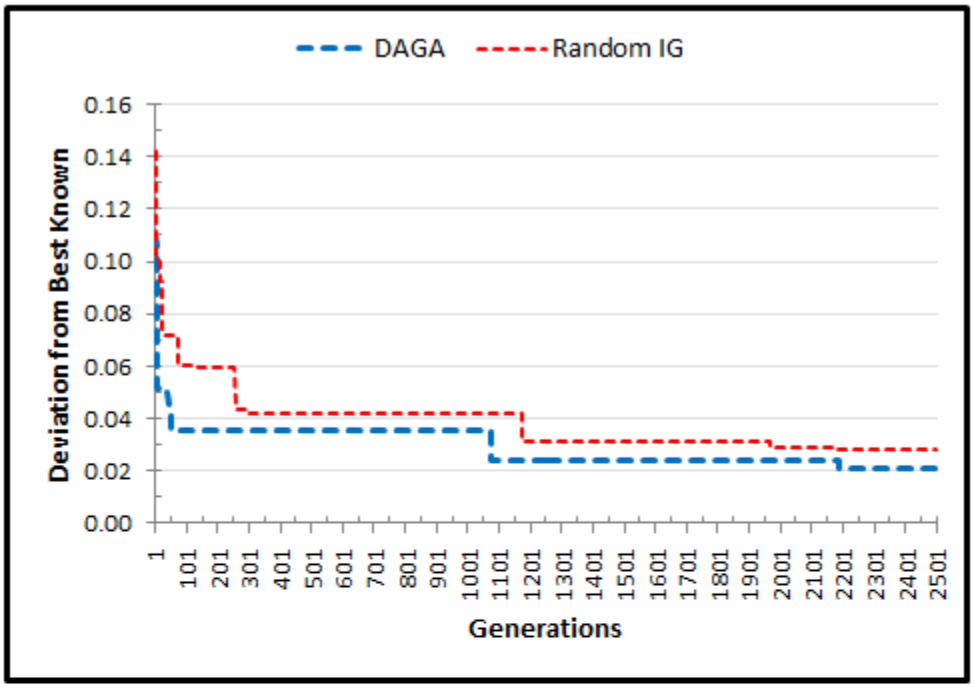
In summary it was concluded that using a structured initial population had no significant impact on DAGA's ability to find an optimal solution but that it did create a better initial solution and allowed the algorithm to perform better early in the search and produce a relatively good solution early in the search. The domain aware crossover operator produced superior solutions and had a significant impact on the overall functionality of DAGA. The domain aware mutation operator did not have a large impact

on the overall functionality of DAGA and may be inferior to a random approach to mutation.

Lastly it can be concluded that a genetic algorithm that incorporates domain knowledge into its design can have a positive impact on its ability to find optimal solutions for the *p*-median problem. This conclusion adds to the body of knowledge about genetic algorithms and the *p*-median problem and could serve as a basis for further research on the topic.

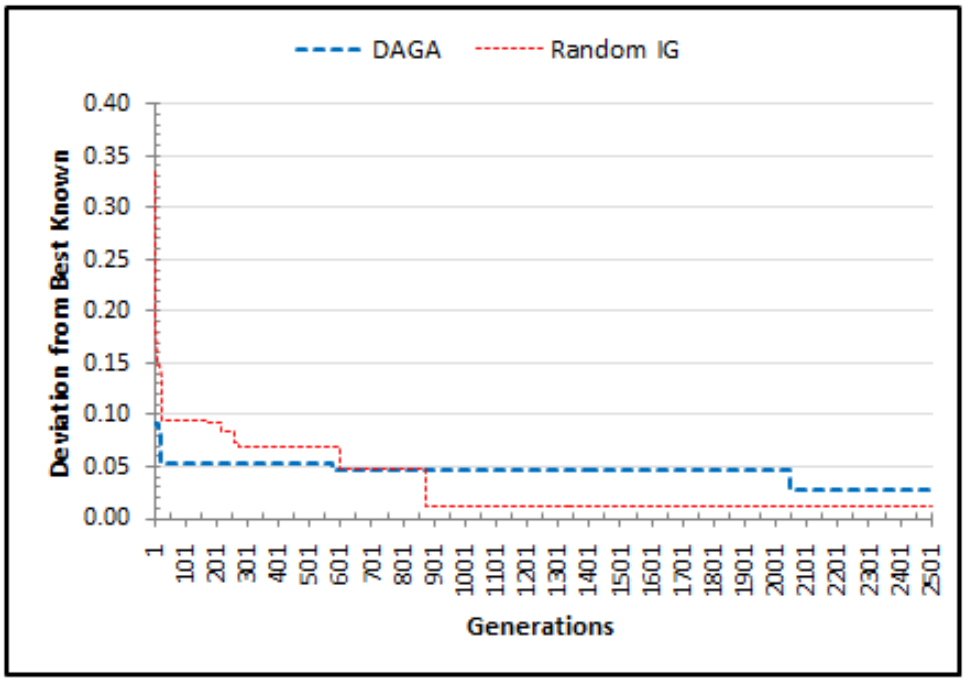
Appendix A

Run Profiles Comparing DAGA and Random Initial Generation



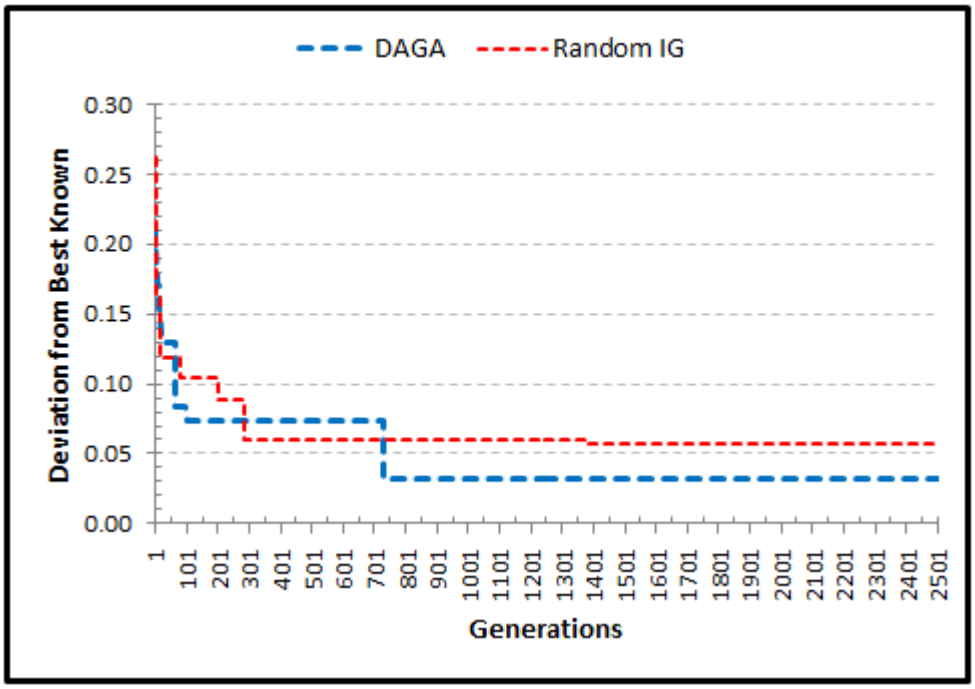
DAGA			Random IG		
Gen	FitVal	Err	Gen	FitVal	Err
1	112,412.12	0.11	1	115,638.30	0.14
4	111,980.37	0.11	2	111,413.01	0.10
7	106,451.62	0.05	19	110,602.86	0.09
15	106,285.68	0.05	21	108,553.90	0.07
43	105,843.16	0.05	27	108,493.54	0.07
49	105,612.05	0.04	77	107,355.28	0.06
51	104,862.76	0.04	137	107,259.08	0.06
1077	103,648.44	0.02	248	107,248.20	0.06
2191	103,399.72	0.02	257	105,673.71	0.04
2982	103,359.62	0.02	300	105,458.18	0.04
3094	102,435.59	0.01	1176	104,396.03	0.03
3855	102,421.02	0.01	1966	104,143.88	0.03
5945	101,253.83	0.00	2187	104,116.69	0.03
			4397	101,804.66	0.01

Figure 16. Run profiles with 10 medians



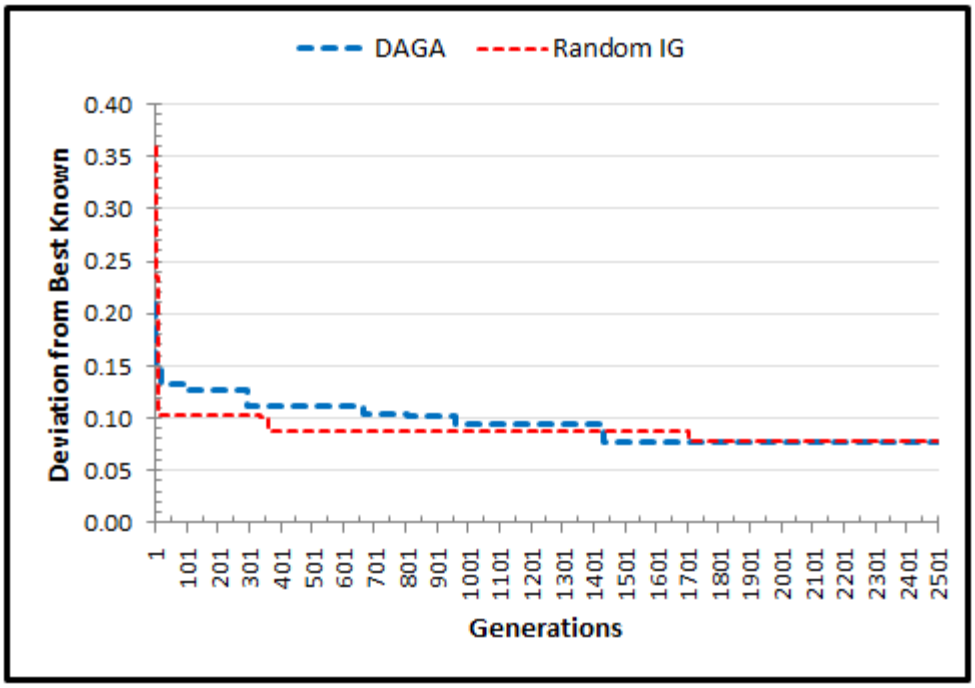
DAGA			Random IG		
Gen	FitVal	Err	Gen	FitVal	Err
1	63,194.03	0.09	1	77,231.75	0.33
12	62,798.86	0.09	2	75,038.75	0.30
24	60,993.44	0.05	3	73,485.29	0.27
573	60,585.90	0.05	4	73,297.11	0.27
2044	59,498.15	0.03	5	73,028.05	0.26
2505	59,295.16	0.02	6	68,531.34	0.18
3042	59,211.94	0.02	7	68,525.62	0.18
3684	58,859.55	0.02	8	66,433.72	0.15
			12	66,382.95	0.15
			22	65,686.32	0.14
			27	63,351.53	0.09
			171	63,312.49	0.09
			174	63,170.46	0.09
			219	62,775.82	0.09
			259	62,159.14	0.07
			273	61,823.87	0.07
			598	60,623.28	0.05
			875	58,543.26	0.01

Figure 17. Run Profiles with 20 Medians



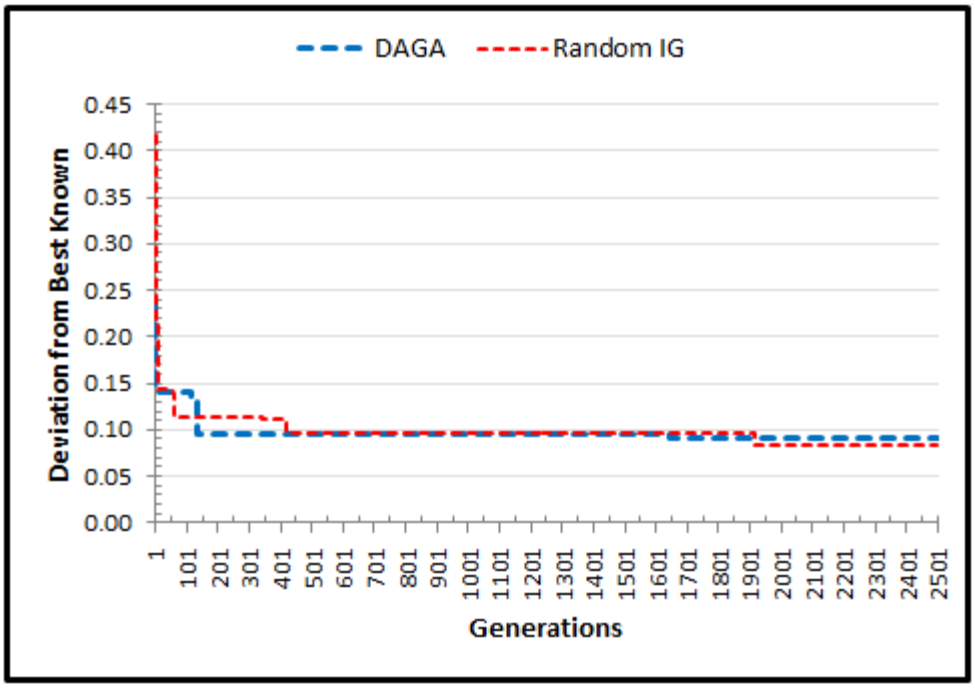
DAGA			Random IG		
Gen	FitVal	Err	Gen	FitVal	Err
1	53,350.54	0.21	1	55,553.49	0.26
3	52,814.07	0.20	4	51,152.51	0.16
4	51,649.83	0.17	16	49,232.39	0.12
7	51,589.14	0.17	79	48,585.85	0.10
11	51,454.67	0.17	202	47,888.06	0.09
14	50,799.90	0.15	283	47,218.66	0.07
17	49,948.89	0.13	284	46,675.68	0.06
22	49,849.17	0.13	1381	46,555.99	0.06
23	49,720.25	0.13	3720	46,425.02	0.05
66	47,667.34	0.08	6177	44,311.24	0.01
100	47,246.43	0.07			
730	45,404.13	0.03			

Figure 18. Run Profiles with 30 Medians



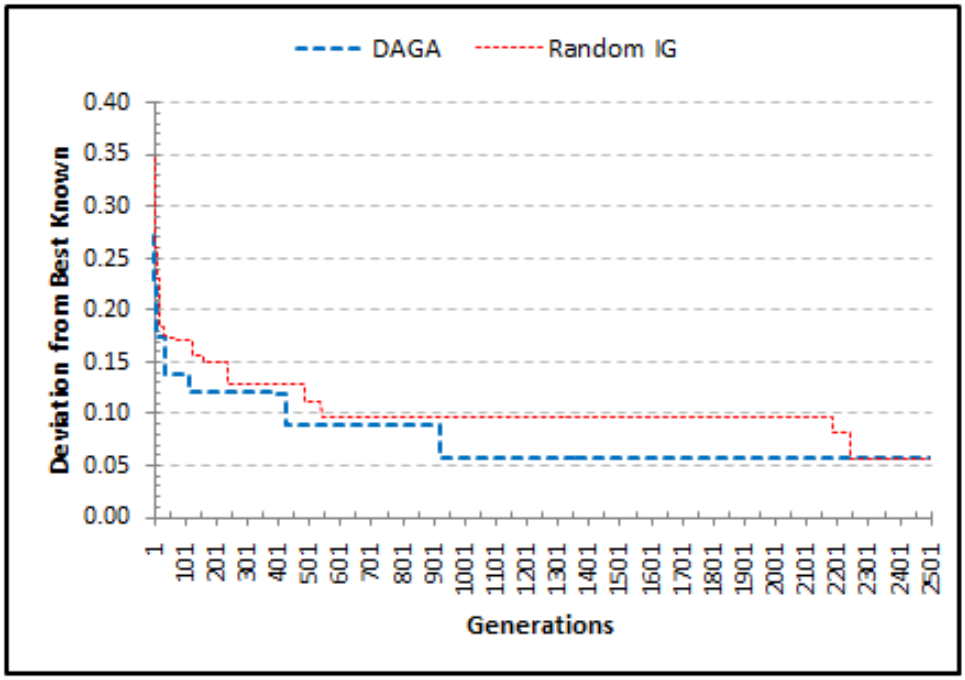
DAGA			Random IG		
Gen	FitVal	Err	Gen	FitVal	Err
1	42,288.72	0.21	1	47,534.19	0.36
2	40,404.06	0.15	2	46,341.07	0.32
19	39,668.81	0.13	3	43,784.38	0.25
88	39,414.57	0.13	6	43,202.31	0.23
296	38,892.94	0.11	9	41,383.75	0.18
634	38,813.12	0.11	13	38,578.79	0.10
667	38,637.25	0.10	335	38,509.85	0.10
807	38,576.39	0.10	365	38,058.95	0.09
961	38,291.63	0.09	1706	37,758.01	0.08
1436	37,698.92	0.08	4032	36,453.38	0.04
2863	37,332.60	0.07			
2866	36,514.57	0.04			

Figure 19. Run Profiles with 40 Medians



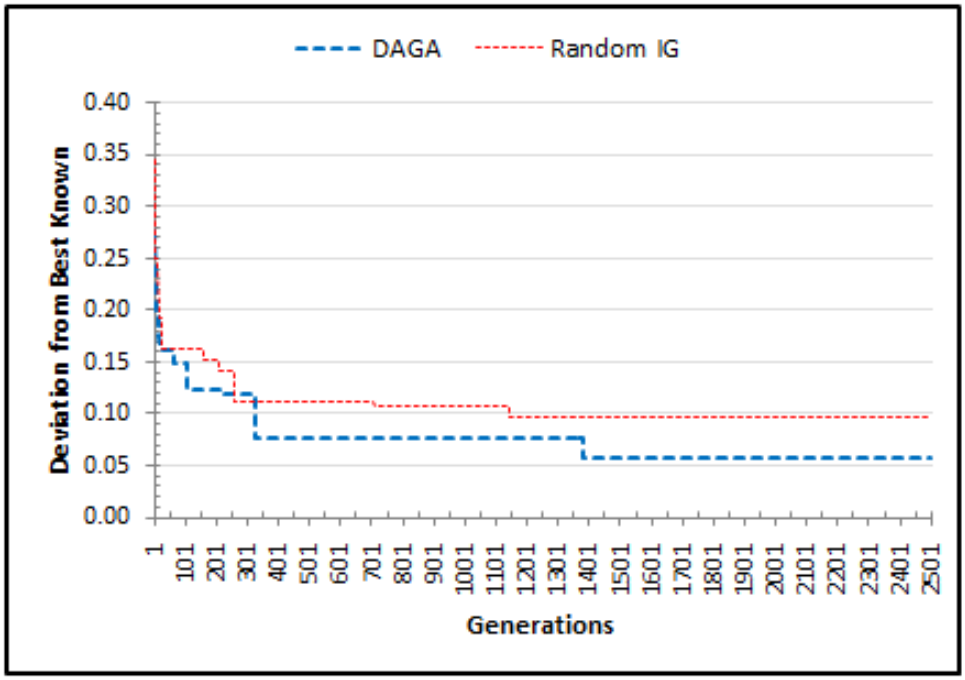
DAGA			Random IG		
Gen	FitVal	Err	Gen	FitVal	Err
1	36,034.07	0.24	1	41,231.43	0.42
2	33,205.37	0.14	2	37,857.02	0.30
44	33,156.23	0.14	4	37,375.30	0.28
115	32,903.04	0.13	6	35,388.52	0.22
136	31,862.85	0.10	12	33,262.40	0.14
1647	31,729.40	0.09	59	33,103.84	0.14
4076	31,486.43	0.08	60	32,415.00	0.11
5813	30,508.74	0.05	341	32,332.40	0.11
7825	30,240.72	0.04	419	31,912.52	0.10
			918	31,866.90	0.10
			1918	31,513.62	0.08
			3585	31,406.09	0.08
			4561	31,080.68	0.07

Figure 20. Run Profiles with 50 Medians



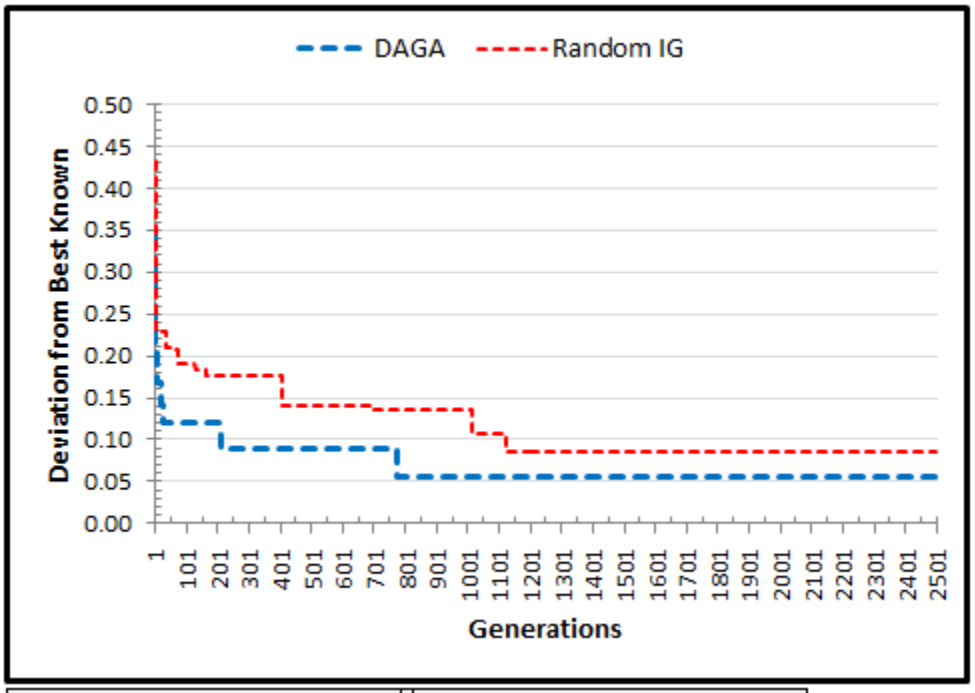
DAGA			Random IG		
Gen	FitVal	Err	Gen	FitVal	Err
1	31,998.31	0.27	1	33,897.61	0.35
3	30,950.21	0.23	3	32,724.06	0.30
5	30,387.54	0.21	5	32,162.30	0.28
7	29,566.26	0.18	7	31,690.46	0.26
33	28,620.62	0.14	12	31,596.51	0.26
111	28,199.55	0.12	13	30,953.99	0.23
381	28,140.07	0.12	16	30,321.56	0.21
424	27,433.28	0.09	17	29,800.53	0.18
919	26,620.11	0.06	32	29,594.57	0.18
			50	29,539.94	0.17
			67	29,460.73	0.17
			124	29,072.25	0.16
			162	28,928.66	0.15
			237	28,381.43	0.13
			486	27,982.22	0.11
			539	27,571.96	0.10
			2185	27,201.21	0.08
			2241	26,588.45	0.06

Figure 21. Run Profiles with 60 Medians



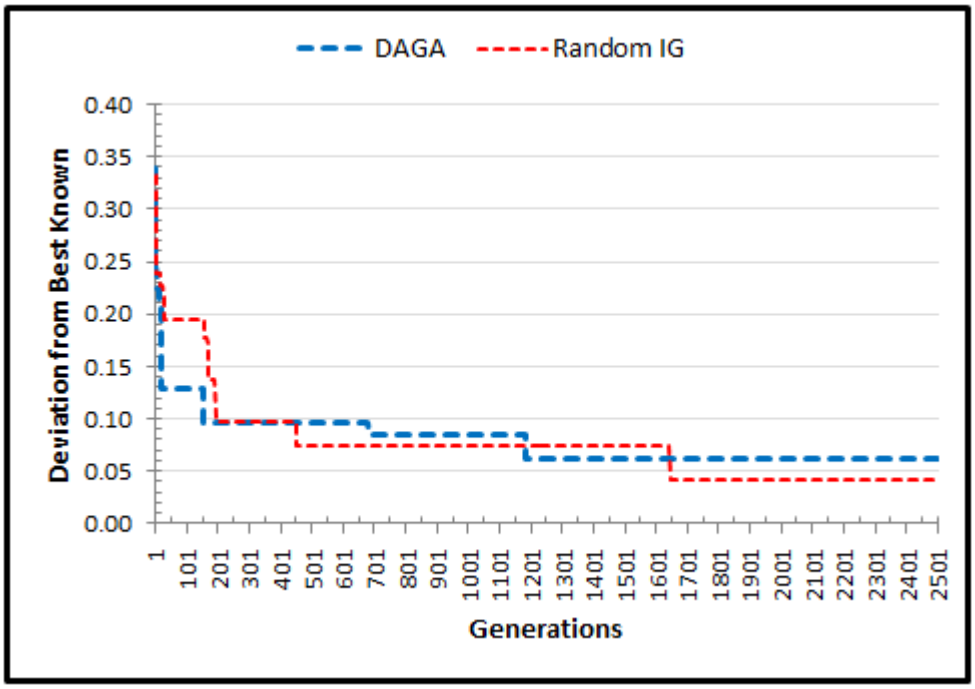
DAGA			Random IG		
Gen	FitVal	Err	Gen	FitVal	Err
1	28,235.12	0.28	1	29,768.21	0.35
3	28,140.90	0.27	2	29,364.18	0.33
4	26,908.17	0.22	3	29,033.23	0.31
5	26,476.83	0.20	5	28,625.25	0.29
15	25,992.45	0.17	7	27,636.43	0.25
18	25,719.45	0.16	10	27,619.13	0.25
66	25,398.36	0.15	14	27,408.79	0.24
109	24,839.12	0.12	15	26,390.01	0.19
213	24,746.70	0.12	24	25,723.38	0.16
326	23,836.25	0.08	36	25,721.30	0.16
1384	23,412.64	0.06	157	25,479.35	0.15
			210	25,242.13	0.14
			261	24,575.33	0.11
			709	24,515.96	0.11
			1105	24,477.71	0.11
			1142	24,245.13	0.10
			3329	23,947.32	0.08
			4518	23,785.69	0.08
			5251	23,578.34	0.07

Figure 22. Run Profiles with 70 medians



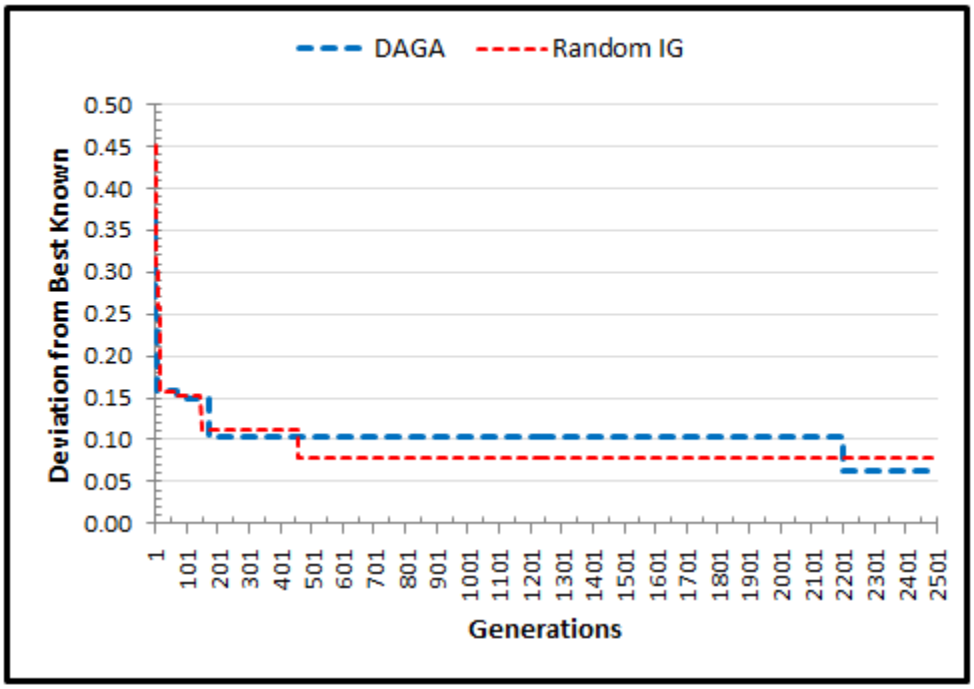
DAGA			Random IG		
Gen	FitVal	Err	Gen	FitVal	Err
1	26,702.36	0.34	1	28,475.79	0.43
2	25,580.31	0.29	2	27,385.61	0.38
3	23,978.62	0.21	3	27,180.75	0.37
9	23,566.91	0.19	4	25,131.67	0.26
10	23,198.30	0.17	6	24,398.95	0.23
21	22,666.42	0.14	34	24,335.00	0.22
28	22,242.63	0.12	38	24,039.14	0.21
209	21,942.89	0.10	69	23,993.04	0.21
212	21,626.11	0.09	75	23,646.83	0.19
776	20,958.67	0.05	129	23,504.85	0.18
			166	23,362.33	0.18
			406	22,643.71	0.14
			700	22,589.38	0.14
			1013	22,018.25	0.11
			1126	21,571.70	0.09

Figure 23. Run Profiles for 80 Medians



DAGA			Random IG		
Gen	FitVal	Err	Gen	FitVal	Err
1	24,087.98	0.34	1	23,965.82	0.33
2	22,410.75	0.25	4	23,822.31	0.32
4	22,229.17	0.24	5	22,271.46	0.24
7	22,019.03	0.22	17	22,083.93	0.23
13	21,829.85	0.21	25	22,072.25	0.23
19	20,881.40	0.16	27	21,914.06	0.22
20	20,309.57	0.13	31	21,487.16	0.19
155	19,711.91	0.10	157	21,184.27	0.18
682	19,524.13	0.09	167	21,075.12	0.17
1182	19,085.52	0.06	168	20,450.11	0.14
			193	19,735.93	0.10
			452	19,581.71	0.09
			453	19,320.41	0.07
			1365	19,320.24	0.07
			1645	18,736.86	0.04

Figure 24. Run Profile with 90 Medians



DAGA			Random IG		
Gen	FitVal	Err	Gen	FitVal	Err
1	22,694.86	0.37	1	24,034.30	0.45
2	21,776.22	0.32	2	23,309.76	0.41
3	20,649.70	0.25	3	21,601.15	0.31
7	19,186.55	0.16	11	20,825.69	0.26
72	19,048.75	0.15	16	20,706.78	0.25
104	19,001.81	0.15	19	19,159.18	0.16
171	18,985.81	0.15	69	19,098.15	0.15
172	18,261.34	0.10	70	19,066.80	0.15
2204	17,580.43	0.06	148	18,404.62	0.11
			460	17,850.18	0.08

Figure 25. Run Profiles with 100 Medians

Appendix B

Detailed Results

Problem Set fl1400; Crossover Technique 1

Median Count = 10		
Run	**Result**	**Gens**
1	102,416.28	7,753
2	101,248.57	8,446
3	101,748.98	3,131
4	101,714.52	5,579
5	102,598.73	5,568
6	102,115.40	7,588
7	102,691.40	3,458
8	102,113.00	3,650
9	102,125.48	3,752
10	102,711.90	3,988
Avg	102,148.43	5,291
Min	101,248.57	3,131
Max	102,711.90	8,446

Median Count = 20		
Run	**Result**	**Gens**
1	59,908.74	4,747
2	59,049.09	3,144
3	58,953.50	6,471
4	58,859.55	6,185
5	60,410.66	2,527
6	59,310.37	6,655
7	59,574.13	2,567
8	59,653.22	2,672
9	60,449.35	2,626
10	59,833.71	8,979
Avg	59,600.23	2,527
Min	58,859.55	8,979
Max	60,449.35	4,657

Median Count = 30		
Run	**Result**	**Gens**
1	46,294.18	3,471
2	47,011.97	3,220
3	46,452.08	3,516
4	45,459.94	4,395
5	47,729.94	3,782
6	46,915.72	5,010
7	46,857.27	2,587
8	46,538.94	8,542
9	45,404.13	3,231
10	46,109.16	6,703
Avg	46,477.33	4,446
Min	45,404.13	2,587
Max	47,729.94	8,542

Median Count = 40		
Run	**Result**	**Gens**
1	36,514.57	5,367
2	37,252.23	6,556
3	36,915.19	4,589
4	37,741.65	2,722
5	36,541.71	4,799
6	37,178.68	3,099
7	37,217.61	6,728
8	36,932.81	5,222
9	37,713.41	7,297
10	36,938.67	4,825
Avg	37,094.65	5,120
Min	36,514.57	2,722
Max	37,741.65	7,297

Median Count = 50		
Run	**Result**	**Gens**
1	31,262.88	3,917
2	30,598.26	6,854
3	31,185.95	3,800
4	31,156.81	5,371
5	30,240.72	10,326
6	30,883.15	4,972
7	31,168.04	2,674
8	31,059.10	2,523
9	30,442.90	4,501
10	30,835.94	3,179
Avg	30,883.38	4,812
Min	30,240.72	2,523
Max	31,262.88	10,326

Median Count = 60		
Run	**Result**	**Gens**
1	27,124.27	4,183
2	27,445.95	2,600
3	27,313.02	7,163
4	27,167.93	6,437
5	27,413.05	6,080
6	26,620.11	3,420
7	27,270.67	3,776
8	26,892.97	3,908
9	27,682.85	5,060
10	27,113.02	4,104
Avg	27,204.38	4,673
Min	26,620.11	2,600
Max	27,682.85	7,163

Median Count = 70		
Run	**Result**	**Gens**
1	23,897.25	7,060
2	23,809.97	3,861
3	24,225.52	4,342
4	23,412.64	3,885
5	24,249.13	4,914
6	23,813.76	5,879
7	23,750.74	3,205
8	24,307.24	6,456
9	24,008.39	4,620
10	24,869.24	3,423
Avg	24,034.39	4,765
Min	23,412.64	3,205
Max	24,869.24	7,060

Median Count = 80		
Run	**Result**	**Gens**
1	22,043.04	4,889
2	22,280.45	6,953
3	21,262.07	2,582
4	22,071.81	4,266
5	21,821.51	5,757
6	22,107.23	2,696
7	21,092.20	4,616
8	20,958.67	3,277
9	21,111.85	5,453
10	21,894.17	2,546
Avg	21,664.30	4,304
Min	20,958.67	2,546
Max	22,280.45	6,953

Median Count = 90		
Run	**Result**	**Gens**
1	19,858.59	2,593
2	20,025.31	2,746
3	19,384.50	6,052
4	19,298.61	7,599
5	19,539.11	2,843
6	19,232.87	3,031
7	19,085.52	3,683
8	19,740.22	3,090
9	19,285.53	5,949
10	19,837.69	2,522
Avg	19,528.80	4,011
Min	19,085.52	2,522
Max	20,025.31	7,599

Median Count = 100		
Run	**Result**	**Gens**
1	18,228.66	3,974
2	18,143.40	3,626
3	18,391.17	4,568
4	18,262.19	2,549
5	17,854.83	3,956
6	18,423.08	2,545
7	17,580.43	4,705
8	18,258.07	2,895
9	18,295.52	4,440
10	17,853.07	3,128
Avg	18,129.04	3,639
Min	17,580.43	2,545
Max	18,423.08	4,705

Problem Set fl1400; Crossover Technique 2

Median Count = 10		
Run	**Result**	**Gens**
1	106,376.88	4,300
2	104,608.10	3,877
3	104,397.46	4,169
4	103,260.86	3,311
5	105,173.47	3,084
6	105,804.78	5,132
7	105,327.98	2,585
8	105,178.68	5,418
9	105,774.07	3,548
10	106,808.55	4,272
Avg	105,271.08	3,970
Min	103,260.86	2,585
Max	106,808.55	5,418

Median Count = 20		
Run	**Result**	**Gens**
1	61,561.51	4,747
2	60,211.52	3,144
3	59,620.89	6,471
4	60,579.14	6,185
5	61,913.57	2,527
6	60,627.72	6,655
7	60,417.14	2,567
8	60,905.44	2,672
9	62,147.94	2,626
10	60,748.53	8,979
Avg	60,873.34	2,527
Min	59,620.89	8,979
Max	62,147.94	4,657

Median Count = 30		
Run	**Result**	**Gens**
1	47,327.61	3,471
2	48,718.19	3,220
3	47,662.64	3,516
4	46,938.20	4,395
5	49,152.84	3,782
6	48,312.90	5,010
7	48,440.89	2,587
8	48,092.35	8,542
9	46,408.24	3,231
10	47,202.12	6,703
Avg	47,825.60	4,446
Min	46,408.24	2,587
Max	49,152.84	8,542

Median Count = 40		
Run	**Result**	**Gens**
1	37,382.30	5,367
2	38,246.89	6,556
3	37,679.88	4,589
4	38,711.76	2,722
5	37,624.28	4,799
6	38,150.00	3,099
7	38,139.85	6,728
8	37,677.86	5,222
9	38,803.48	7,297
10	37,757.32	4,825
Avg	38,017.36	5,120
Min	37,382.30	2,722
Max	38,803.48	7,297

Median Count = 50		
Run	**Result**	**Gens**
1	31,910.26	3,917
2	31,590.55	6,854
3	32,105.09	3,800
4	32,093.11	5,371
5	31,233.34	10,326
6	31,539.23	4,972
7	32,246.68	2,674
8	31,719.94	2,523
9	31,390.06	4,501
10	31,456.63	3,179
Avg	31,728.49	4,812
Min	31,233.34	2,523
Max	32,246.68	10,326

Median Count = 60		
Run	**Result**	**Gens**
1	27,818.13	4,183
2	28,309.14	2,600
3	28,316.73	7,163
4	28,101.99	6,437
5	28,185.60	6,080
6	26,923.56	3,420
7	28,081.20	3,776
8	27,420.30	3,908
9	28,724.73	5,060
10	27,397.21	4,104
Avg	27,927.86	4,673
Min	26,923.56	2,600
Max	28,724.73	7,163

Median Count = 70		
Run	**Result**	**Gens**
1	24,446.64	7,060
2	24,499.57	3,861
3	24,599.44	4,342
4	23,877.50	3,885
5	24,548.08	4,914
6	24,428.74	5,879
7	24,114.64	3,205
8	24,997.08	6,456
9	24,458.80	4,620
10	25,255.08	3,423
Avg	24,522.56	4,765
Min	23,877.50	3,205
Max	25,255.08	7,060

Median Count = 80		
Run	**Result**	**Gens**
1	22,316.96	4,889
2	22,544.75	6,953
3	21,575.56	2,582
4	22,348.67	4,266
5	22,310.17	5,757
6	22,365.82	2,696
7	21,696.41	4,616
8	21,345.60	3,277
9	21,425.85	5,453
10	22,484.49	2,546
Avg	22,041.43	4,304
Min	21,345.60	2,546
Max	22,544.75	6,953

Median Count = 90		
Run	**Result**	**Gens**
1	20,222.28	2,593
2	20,575.70	2,746
3	19,948.97	6,052
4	19,939.66	7,599
5	20,134.52	2,843
6	19,731.79	3,031
7	19,378.26	3,683
8	20,253.29	3,090
9	19,850.93	5,949
10	20,220.71	2,522
Avg	20,025.61	4,011
Min	19,378.26	2,522
Max	20,575.70	7,599

Median Count = 100		
Run	**Result**	**Gens**
1	18,634.50	3,974
2	18,815.66	3,626
3	19,086.31	4,568
4	18,622.26	2,549
5	18,085.81	3,956
6	18,695.60	2,545
7	18,000.88	4,705
8	18,778.39	2,895
9	18,686.85	4,440
10	18,551.61	3,128
Avg	18,595.79	3,639
Min	18,000.88	2,545
Max	19,086.31	4,705

Problem Set pcb3038; Crossover Technique 1

Median Count = 10		
Run	**Result**	**Gens**
1	1,235,943.75	7,349
2	1,236,644.69	4,844
3	1,235,657.95	4,271
4	1,239,459.45	3,620
5	1,256,273.85	6,992
6	1,260,371.11	3,311
7	1,251,368.79	3,242
8	1,250,643.29	8,388
9	1,251,029.89	8,971
10	1,258,174.29	4,246
Avg	1,247,556.71	5,523
Min	1,235,657.95	3,242
Max	1,260,371.11	8,971

Median Count = 20		
Run	**Result**	**Gens**
1	881,001.32	3,763
2	872,276.07	2,887
3	872,897.78	8,523
4	869,815.06	8,039
5	880,212.09	4,428
6	878,056.15	8,287
7	873,545.67	4,400
8	881,377.98	8,331
9	872,133.96	3,109
10	866,207.93	8,338
Avg	874,752.40	6,011
Min	866,207.93	2,887
Max	881,377.98	8,523

Median Count = 30		
Run	**Result**	**Gens**
1	708,131.39	4,569
2	703,159.20	4,996
3	713,591.45	8,199
4	701,283.38	6,368
5	702,484.12	4,591
6	704,195.25	3,005
7	705,402.82	4,076
8	716,307.87	3,879
9	704,780.89	8,065
10	719,511.81	8,639
Avg	707,884.82	5,639
Min	701,283.38	3,005
Max	719,511.81	8,639

Median Count = 40		
Run	**Result**	**Gens**
1	605,612.20	5,076
2	607,961.80	5,910
3	595,626.06	4,810
4	604,377.57	5,764
5	602,560.20	5,755
6	598,796.24	7,223
7	607,097.69	5,014
8	596,916.46	3,637
9	605,945.50	6,141
10	595,706.25	6,167
Avg	602,060.00	5,550
Min	595,626.06	3,637
Max	607,961.80	7,223

Median Count = 50		
Run	**Result**	**Gens**
1	537,317.86	3,324
2	538,062.02	6,068
3	536,445.73	3,714
4	529,623.54	6,830
5	531,807.30	7,167
6	539,099.21	7,037
7	532,779.82	8,244
8	532,142.90	3,449
9	538,278.41	8,626
10	538,947.10	4,351
Avg	535,450.39	5,881
Min	529,623.54	3,324
Max	539,099.21	8,626

Median Count = 60		
Run	**Result**	**Gens**
1	493,149.64	5,068
2	491,572.44	2,946
3	489,638.62	5,231
4	489,833.93	8,133
5	494,878.73	4,764
6	485,164.14	2,717
7	487,738.42	3,353
8	491,927.27	5,533
9	493,085.21	8,124
10	484,576.92	7,748
Avg	490,156.53	5,362
Min	484,576.92	2,717
Max	494,878.73	8,133

Median Count = 70		
Run	**Result**	**Gens**
1	452,394.29	4,285
2	454,023.38	2,950
3	452,333.41	8,310
4	453,860.52	5,335
5	454,227.90	4,258
6	448,855.74	8,114
7	453,952.48	7,506
8	448,061.43	7,584
9	457,397.02	7,910
10	453,703.51	2,813
Avg	452,880.97	5,907
Min	448,061.43	2,813
Max	457,397.02	8,310

Median Count = 80		
Run	**Result**	**Gens**
1	420,806.57	6,462
2	429,052.31	7,696
3	422,099.61	4,143
4	423,004.84	4,235
5	430,868.60	3,973
6	421,384.67	6,622
7	423,790.31	6,203
8	427,586.78	6,277
9	419,612.42	3,317
10	427,788.43	4,168
Avg	424,599.45	5,310
Min	419,612.42	3,317
Max	430,868.60	7,696

Median Count = 90		
Run	**Result**	**Gens**
1	404,771.33	6,456
2	406,429.04	3,712
3	396,657.80	8,584
4	404,067.64	8,657
5	402,788.73	4,443
6	397,256.70	4,461
7	403,402.94	8,505
8	396,916.65	7,264
9	398,881.09	8,705
10	401,961.41	6,144
Avg	401,313.33	6,693
Min	396,657.80	3,712
Max	406,429.04	8,705

Median Count = 100		
Run	**Result**	**Gens**
1	386,672.87	5,007
2	386,551.44	3,882
3	384,048.27	8,022
4	384,959.78	8,654
5	381,249.90	5,136
6	387,810.62	8,526
7	384,920.82	7,731
8	380,153.39	6,295
9	382,691.54	8,685
10	382,837.59	5,285
Avg	384,189.62	6,722
Min	380,153.39	3,882
Max	387,810.62	8,685

Problem Set pcb3038; Crossover Technique 2

Median Count = 10		
Run	**Result**	**Gens**
1	1,283,539.71	3,733
2	1,263,195.04	3,600
3	1,260,629.99	8,889
4	1,265,596.75	4,864
5	1,297,922.89	2,701
6	1,289,843.80	3,584
7	1,282,179.46	2,801
8	1,289,450.86	2,677
9	1,286,664.36	3,959
10	1,287,308.57	4,399
Avg	1,280,633.14	4,121
Min	1,260,629.99	2,677
Max	1,297,922.89	8,889

Median Count = 20		
Run	**Result**	**Gens**
1	916,036.39	4,937
2	897,582.64	3,175
3	890,561.86	7,118
4	893,303.74	6,309
5	900,240.03	4,181
6	896,844.67	6,855
7	893,561.25	3,888
8	906,947.31	3,953
9	893,548.85	3,807
10	896,042.66	8,530
Avg	898,466.94	3,175
Min	890,561.86	8,530
Max	916,036.39	5,275

Median Count = 30		
Run	**Result**	**Gens**
1	731,883.91	3,714
2	718,664.68	3,478
3	728,478.36	3,832
4	726,166.95	4,483
5	720,736.27	3,631
6	722,471.16	4,860
7	733,257.81	3,551
8	735,138.19	8,713
9	726,925.92	2,973
10	739,512.88	7,038
Avg	728,323.61	4,627
Min	718,664.68	2,973
Max	739,512.88	8,713

Median Count = 40		
Run	**Result**	**Gens**
1	621,275.00	4,991
2	627,555.51	7,212
3	610,078.42	4,635
4	625,904.62	4,020
5	623,727.17	5,279
6	620,228.75	2,882
7	619,430.35	6,593
8	620,541.81	5,013
9	627,428.17	7,662
10	610,342.55	5,211
Avg	620,651.24	5,350
Min	610,078.42	2,882
Max	627,555.51	7,662

Median Count = 50		
Run	**Result**	**Gens**
1	558,103.42	3,525
2	556,422.80	6,648
3	548,105.61	3,572
4	543,697.60	5,532
5	550,891.11	9,913
6	559,375.54	5,022
7	547,476.04	2,701
8	548,630.55	3,840
9	558,077.78	4,186
10	552,114.57	3,274
Avg	552,289.50	4,821
Min	543,697.60	2,701
Max	559,375.54	9,913

Median Count = 60		
Run	**Result**	**Gens**
1	505,020.48	4,308
2	504,432.55	3,340
3	503,289.98	7,736
4	501,153.87	6,115
5	511,506.81	6,506
6	495,332.18	3,146
7	500,512.84	3,398
8	508,656.00	4,064
9	504,664.14	4,908
10	500,892.87	4,145
Avg	503,546.17	4,767
Min	495,332.18	3,146
Max	511,506.81	7,736

Median Count = 70		
Run	**Result**	**Gens**
1	467,006.90	7,554
2	466,353.44	4,247
3	469,918.63	3,951
4	466,662.43	4,079
5	471,806.97	4,619
6	462,914.87	5,820
7	463,664.50	3,333
8	463,570.73	6,004
9	466,798.50	4,851
10	466,993.23	3,526
Avg	466,569.02	4,798
Min	462,914.87	3,333
Max	471,806.97	7,554

Median Count = 80		
Run	**Result**	**Gens**
1	435,805.89	5,329
2	438,982.68	6,466
3	432,562.96	3,505
4	435,397.00	4,437
5	445,129.31	5,987
6	435,657.76	2,723
7	435,287.34	4,708
8	442,865.73	3,048
9	433,389.69	5,889
10	437,059.49	2,699
Avg	437,213.79	4,479
Min	432,562.96	2,699
Max	445,129.31	6,466

Median Count = 90		
Run	**Result**	**Gens**
1	417,415.13	3,737
2	420,172.50	4,008
3	408,568.76	6,173
4	418,261.21	8,359
5	417,855.74	3,070
6	407,760.93	3,122
7	419,528.28	3,978
8	410,330.32	3,245
9	407,519.87	5,414
10	414,676.87	3,874
Avg	414,208.96	4,498
Min	407,519.87	3,070
Max	420,172.50	8,359

Median Count = 100		
Run	**Result**	**Gens**
1	397,824.02	4,252
2	402,003.86	3,263
3	393,160.99	4,796
4	396,686.19	3,443
5	396,233.41	4,312
6	397,901.41	3,474
7	395,370.87	4,799
8	388,203.24	3,973
9	396,372.87	4,662
10	394,803.34	3,097
Avg	395,856.02	4,007
Min	388,203.24	3,097
Max	402,003.86	4,799

Problem Set rl5934; Crossover Technique 1

Median Count = 10		
Run	**Result**	**Gens**
1	10,135,743.17	7,725
2	10,147,346.07	3,773
3	10,017,228.62	4,745
4	10,023,073.42	8,454
5	10,038,707.84	8,784
6	10,141,451.24	5,151
7	9,948,378.50	3,412
8	10,093,124.13	5,373
9	10,052,208.96	7,483
10	10,115,164.45	7,121
Avg	10,071,242.64	6,202
Min	9,948,378.50	3,412
Max	10,147,346.07	8,784

Median Count = 20		
Run	**Result**	**Gens**
1	7,022,427.34	6,638
2	6,935,624.99	2,813
3	7,056,057.80	3,838
4	7,031,889.77	8,386
5	6,931,397.86	4,716
6	7,040,245.50	4,237
7	7,018,171.81	5,799
8	7,037,020.90	4,379
9	6,937,278.47	4,521
10	7,000,761.28	3,271
Avg	7,001,087.57	4,860
Min	6,931,397.86	2,813
Max	7,056,057.80	8,386

Median Count = 30		
Run	**Result**	**Gens**
1	5,675,255.81	4,904
2	5,694,607.61	2,582
3	5,749,487.43	3,927
4	5,676,644.20	8,325
5	5,746,864.17	8,109
6	5,682,416.33	8,032
7	5,621,758.91	4,412
8	5,674,074.70	3,301
9	5,624,744.13	8,386
10	5,719,771.07	7,030
Avg	5,686,562.44	5,901
Min	5,621,758.91	2,582
Max	5,749,487.43	8,386

Median Count = 40		
Run	**Result**	**Gens**
1	4,822,296.41	5,204
2	4,826,529.01	2,972
3	4,788,835.20	7,468
4	4,808,584.53	4,625
5	4,823,494.14	4,494
6	4,810,927.11	5,160
7	4,841,823.88	5,469
8	4,840,666.42	7,456
9	4,855,561.35	3,129
10	4,861,491.78	2,973
Avg	4,828,020.98	4,895
Min	4,788,835.20	2,972
Max	4,861,491.78	7,468

Median Count = 50		
Run	**Result**	**Gens**
1	4,254,430.87	3,001
2	4,291,536.54	2,581
3	4,291,109.94	4,424
4	4,250,951.44	7,992
5	4,308,526.23	7,376
6	4,252,180.13	6,307
7	4,267,464.59	7,473
8	4,278,443.40	3,669
9	4,227,396.73	7,415
10	4,270,699.78	7,783
Avg	4,269,273.97	5,802
Min	4,227,396.73	2,581
Max	4,308,526.23	7,992

Median Count = 60		
Run	**Result**	**Gens**
1	3,887,213.20	3,624
2	3,853,871.60	3,458
3	3,924,787.17	5,259
4	3,859,898.17	5,437
5	3,853,663.06	5,178
6	3,851,771.38	7,439
7	3,850,777.76	6,764
8	3,917,798.22	7,466
9	3,843,454.42	6,372
10	3,913,730.34	3,565
Avg	3,875,696.53	5,456
Min	3,843,454.42	3,458
Max	3,924,787.17	7,466

Median Count = 70		
Run	**Result**	**Gens**
1	3,611,144.13	6,912
2	3,538,947.96	4,522
3	3,594,246.00	4,215
4	3,612,279.31	5,458
5	3,541,848.48	8,267
6	3,585,199.34	5,985
7	3,565,983.42	8,420
8	3,555,995.74	3,685
9	3,599,829.82	5,714
10	3,539,820.17	3,284
Avg	3,574,529.44	5,646
Min	3,538,947.96	3,284
Max	3,612,279.31	8,420

Median Count = 80		
Run	**Result**	**Gens**
1	3,282,953.12	8,682
2	3,297,227.19	4,185
3	3,347,632.73	8,679
4	3,359,261.88	5,347
5	3,314,409.10	7,343
6	3,342,338.94	6,547
7	3,348,636.63	7,735
8	3,354,348.45	3,582
9	3,367,070.46	5,907
10	3,350,200.91	6,364
Avg	3,336,407.94	6,437
Min	3,282,953.12	3,582
Max	3,367,070.46	8,682

Median Count = 90		
Run	**Result**	**Gens**
1	3,111,789.73	2,991
2	3,100,935.25	7,648
3	3,153,436.64	8,798
4	3,127,580.82	7,059
5	3,146,745.81	6,098
6	3,101,913.85	5,765
7	3,149,511.12	6,433
8	3,125,306.40	7,707
9	3,133,328.93	9,070
10	3,090,483.72	3,295
Avg	3,124,103.23	6,486
Min	3,090,483.72	2,991
Max	3,153,436.64	9,070

Median Count = 100		
Run	**Result**	**Gens**
1	2,928,355.00	3,130
2	2,932,922.03	3,134
3	2,939,391.58	8,391
4	2,949,947.43	7,806
5	2,968,394.00	5,245
6	2,965,612.96	3,793
7	2,959,730.98	8,768
8	2,951,770.37	6,493
9	3,000,827.73	8,300
10	2,925,863.34	7,853
Avg	2,952,281.54	6,291
Min	2,925,863.34	3,130
Max	3,000,827.73	8,768

Problem Set rl5934; Crossover Technique 2

Median Count = 10		
Run	**Result**	**Gens**
1	10,466,550.99	3,658
2	10,421,759.16	3,708
3	10,386,481.03	9,156
4	10,332,059.39	5,107
5	10,337,783.55	3,812
6	10,421,964.25	3,799
7	10,209,378.78	3,081
8	10,311,634.57	3,934
9	10,281,512.97	4,157
10	10,318,146.70	4,707
Avg	10,348,727.14	4,512
Min	10,209,378.78	3,081
Max	10,466,550.99	9,156

Median Count = 20		
Run	**Result**	**Gens**
1	7,234,528.17	4,367
2	7,139,749.64	3,396
3	7,336,289.91	6,406
4	7,234,854.79	5,876
5	7,078,046.08	3,386
6	7,302,374.19	6,722
7	7,205,545.50	3,246
8	7,267,286.95	3,856
9	7,199,834.99	3,699
10	7,154,456.84	9,518
Avg	7,215,296.71	5,047
Min	7,078,046.08	3,246
Max	7,336,289.91	9,518

Median Count = 30		
Run	**Result**	**Gens**
1	5,891,483.43	3,575
2	5,825,801.81	3,445
3	5,902,529.18	3,797
4	5,898,856.20	4,307
5	5,909,611.84	3,895
6	5,891,882.88	5,060
7	5,810,175.13	3,730
8	5,841,283.92	8,371
9	5,843,662.52	2,940
10	5,921,169.79	6,837
Avg	5,873,645.67	4,596
Min	5,810,175.13	2,940
Max	5,921,169.79	8,371

Median Count = 40		
Run	**Result**	**Gens**
1	4,979,944.78	5,904
2	4,946,422.14	6,097
3	4,939,578.00	4,956
4	4,994,837.31	3,834
5	4,976,755.21	4,751
6	4,972,839.65	3,192
7	4,946,024.56	7,132
8	5,022,422.26	5,483
9	5,013,119.96	6,786
10	4,968,735.63	4,487
Avg	4,976,067.95	5,262
Min	4,939,578.00	3,192
Max	5,022,422.26	7,132

Median Count = 50		
Run	**Result**	**Gens**
1	4,341,185.98	4,152
2	4,456,145.69	6,923
3	4,441,930.91	3,496
4	4,361,439.12	5,747
5	4,473,520.48	10,016
6	4,379,258.38	5,320
7	4,375,700.50	3,747
8	4,385,075.25	3,734
9	4,367,997.03	4,456
10	4,394,347.28	3,306
Avg	4,397,660.06	5,090
Min	4,341,185.98	3,306
Max	4,473,520.48	10,016

Median Count = 60		
Run	**Result**	**Gens**
1	4,001,669.17	3,848
2	3,937,454.67	3,312
3	4,055,578.38	6,518
4	3,962,107.77	6,694
5	3,962,551.93	5,837
6	4,000,276.13	3,488
7	3,953,971.51	3,965
8	4,029,930.79	4,299
9	3,959,376.59	4,706
10	4,020,514.28	4,227
Avg	3,988,343.12	4,689
Min	3,937,454.67	3,312
Max	4,055,578.38	6,694

Median Count = 70		
Run	**Result**	**Gens**
1	3,716,257.06	6,566
2	3,661,380.40	3,977
3	3,683,879.03	4,689
4	3,733,648.25	4,040
5	3,637,122.42	5,258
6	3,663,326.01	5,997
7	3,706,696.22	3,333
8	3,667,675.64	6,585
9	3,689,773.51	4,158
10	3,651,703.45	3,389
Avg	3,681,146.20	4,799
Min	3,637,122.42	3,333
Max	3,733,648.25	6,585

Median Count = 80		
Run	**Result**	**Gens**
1	3,366,446.91	5,036
2	3,393,279.55	7,579
3	3,439,988.68	3,367
4	3,438,301.16	3,839
5	3,392,930.30	5,930
6	3,461,086.64	3,955
7	3,446,258.13	5,078
8	3,469,206.70	3,375
9	3,465,324.30	5,671
10	3,458,149.25	3,191
Avg	3,433,097.16	4,702
Min	3,366,446.91	3,191
Max	3,469,206.70	7,579

Median Count = 90		
Run	**Result**	**Gens**
1	3,201,316.10	3,916
2	3,193,375.47	4,008
3	3,267,340.80	6,415
4	3,232,969.52	8,283
5	3,220,047.57	3,958
6	3,167,696.17	3,092
7	3,252,522.10	3,904
8	3,243,432.67	3,337
9	3,223,447.42	6,544
10	3,179,457.37	3,557
Avg	3,218,160.52	4,701
Min	3,167,696.17	3,092
Max	3,267,340.80	8,283

Median Count = 100		
Run	**Result**	**Gens**
1	2,987,061.70	3,577
2	3,001,108.82	3,916
3	3,016,933.71	4,842
4	3,052,934.56	3,797
5	3,059,442.43	3,679
6	3,065,514.48	3,261
7	3,047,075.48	4,940
8	3,036,614.76	4,246
9	3,083,976.22	4,129
10	3,037,206.76	3,222
Avg	3,038,786.89	3,961
Min	2,987,061.70	3,222
Max	3,083,976.22	4,940

Algorithm DAGA-IG; Problem Set fl1400; Crossover Technique 1

Median Count = 10		
Run	**Result**	**Gens**
1	103,737.70	3,871
2	101,804.66	6,898
3	102,795.78	9,429
4	102,507.43	3,835
5	102,967.02	3,603
6	103,265.44	5,916
7	103,092.83	4,405
8	103,022.71	5,034
9	102,179.76	4,837
10	103,745.91	5,746
Avg	102,911.92	5,357
Min	101,804.66	3,603
Max	103,745.91	9,429

Median Count = 20		
Run	**Result**	**Gens**
1	59,981.35	2,763
2	60,049.83	3,988
3	59,726.99	3,543
4	60,070.57	3,681
5	58,543.26	3,376
6	60,473.34	2,617
7	59,493.87	3,514
8	60,491.51	3,486
9	59,826.52	5,885
10	60,076.28	3,410
Avg	59,873.35	2,617
Min	58,543.26	5,885
Max	60,491.51	3,626

Median Count = 30		
Run	**Result**	**Gens**
1	46,437.75	7,125
2	44,311.24	8,678
3	45,933.04	3,623
4	46,303.00	4,219
5	46,407.57	6,633
6	46,623.74	2,746
7	46,711.93	4,989
8	45,700.69	4,602
9	46,825.71	5,716
10	46,838.95	6,057
Avg	46,209.36	5,439
Min	44,311.24	2,746
Max	46,838.95	8,678

Median Count = 40		
Run	**Result**	**Gens**
1	36,453.38	6,533
2	37,470.39	3,432
3	36,454.95	3,500
4	37,409.56	4,848
5	37,705.89	7,593
6	38,408.52	3,866
7	37,182.45	5,483
8	37,109.98	3,579
9	36,568.75	7,399
10	37,302.28	3,007
Avg	37,206.61	4,924
Min	36,453.38	3,007
Max	38,408.52	7,593

Median Count = 50		
Run	**Result**	**Gens**
1	31,080.68	7,062
2	31,685.23	3,816
3	31,375.28	4,404
4	31,442.08	5,348
5	31,578.12	3,562
6	31,656.08	2,774
7	31,804.49	4,600
8	31,664.78	3,624
9	31,513.42	4,966
10	31,893.96	3,778
Avg	31,569.41	4,393
Min	31,080.68	2,774
Max	31,893.96	7,062

Median Count = 60		
Run	**Result**	**Gens**
1	26,733.63	3,830
2	26,744.77	2,890
3	26,806.01	7,324
4	26,989.49	5,671
5	26,588.45	4,724
6	26,885.46	2,590
7	27,606.66	2,902
8	27,102.89	4,489
9	27,450.50	5,362
10	27,111.86	3,497
Avg	27,001.97	4,328
Min	26,588.45	2,590
Max	27,606.66	7,324

Median Count = 70		
Run	**Result**	**Gens**
1	23,858.84	2,658
2	23,657.27	7,045
3	23,829.74	3,862
4	24,164.90	4,820
5	24,016.86	3,611
6	24,462.94	7,022
7	24,465.27	4,390
8	23,932.10	3,678
9	23,578.34	7,752
10	24,294.44	3,768
Avg	24,026.07	4,861
Min	23,578.34	2,658
Max	24,465.27	7,752

Median Count = 80		
Run	**Result**	**Gens**
1	21,571.70	3,627
2	21,695.94	3,224
3	21,726.65	6,113
4	22,523.63	2,697
5	21,981.57	4,148
6	21,985.62	3,230
7	22,045.94	5,742
8	21,840.99	6,817
9	22,013.07	6,016
10	22,184.43	4,713
Avg	21,956.95	4,633
Min	21,571.70	2,697
Max	22,523.63	6,817

Median Count = 90		
Run	Result	Gens
1	19,551.14	4,769
2	19,991.09	4,164
3	20,597.33	3,195
4	19,686.68	4,267
5	19,992.90	4,244
6	19,478.31	5,932
7	18,736.86	4,146
8	19,989.54	4,411
9	19,934.76	3,159
10	19,399.68	5,695
Avg	19,735.83	4,398
Min	18,736.86	3,159
Max	20,597.33	5,932

Median Count = 100		
Run	Result	Gens
1	17,850.18	2,961
2	18,063.57	4,417
3	18,246.14	2,834
4	17,875.37	3,498
5	17,967.33	4,963
6	18,912.10	2,532
7	18,661.00	4,869
8	17,957.88	5,122
9	18,516.34	3,596
10	18,516.78	3,098
Avg	18,256.67	3,789
Min	17,850.18	2,532
Max	18,912.10	5,122

Algorithm DAGA-CO; Problem Set fl1400; Crossover Technique 1

Median Count = 10		
Run	Result	Gens
1	106,320.50	5,125
2	106,588.04	5,899
3	107,799.61	4,096
4	103,794.02	4,289
5	104,356.55	9,264
6	105,408.83	3,044
7	107,700.88	5,006
8	106,947.50	5,437
9	103,112.17	3,500
10	107,440.69	2,840
Avg	105,946.88	4,850
Min	103,112.17	2,840
Max	107,799.61	9,264

Median Count = 20		
Run	Result	Gens
1	62,893.15	6,094
2	61,465.61	5,346
3	62,471.12	5,188
4	62,807.86	3,660
5	64,172.08	5,450
6	63,428.14	2,940
7	62,840.40	3,036
8	63,118.50	6,497
9	62,219.43	2,502
10	63,061.86	3,075
Avg	62,847.82	2,502
Min	61,465.61	6,497
Max	64,172.08	4,379

Median Count = 30		
Run	**Result**	**Gens**
1	48,999.46	5,110
2	49,184.37	4,183
3	48,497.59	4,029
4	50,213.97	2,669
5	49,754.66	2,667
6	49,903.80	4,340
7	50,640.15	2,502
8	49,672.71	4,266
9	50,646.75	6,517
10	49,503.97	7,486
Avg	49,701.74	4,377
Min	48,497.59	2,502
Max	50,646.75	7,486

Median Count = 40		
Run	**Result**	**Gens**
1	39,873.32	2,639
2	40,369.35	4,429
3	38,474.95	5,355
4	39,976.31	8,263
5	39,634.08	6,053
6	40,560.67	2,656
7	39,049.03	5,868
8	39,931.96	4,729
9	40,794.54	3,734
10	39,726.05	6,125
Avg	39,839.03	4,985
Min	38,474.95	2,639
Max	40,794.54	8,263

Median Count = 50		
Run	**Result**	**Gens**
1	35,084.50	3,648
2	33,739.31	2,518
3	33,357.79	3,372
4	34,531.40	3,246
5	33,597.17	4,648
6	34,090.07	6,050
7	34,273.05	4,925
8	33,572.55	5,000
9	33,877.51	6,579
10	33,532.99	4,325
Avg	33,965.63	4,431
Min	33,357.79	2,518
Max	35,084.50	6,579

Median Count = 60		
Run	**Result**	**Gens**
1	29,459.56	4,165
2	30,143.47	5,589
3	30,206.09	3,571
4	30,139.36	5,987
5	29,517.45	4,228
6	29,735.12	8,242
7	29,879.32	3,507
8	30,111.13	2,953
9	29,656.40	5,708
10	28,377.56	3,151
Avg	29,722.55	4,710
Min	28,377.56	2,953
Max	30,206.09	8,242

Median Count = 70		
Run	**Result**	**Gens**
1	25,587.96	3,074
2	26,801.11	2,903
3	26,923.99	3,254
4	26,396.11	3,562
5	25,876.32	5,782
6	26,297.86	2,688
7	26,017.88	3,354
8	26,257.00	2,503
9	26,280.86	3,812
10	26,209.81	5,168
Avg	26,264.89	3,610
Min	25,587.96	2,503
Max	26,923.99	5,782

Median Count = 80		
Run	**Result**	**Gens**
1	24,095.99	4,938
2	24,034.13	4,108
3	24,541.24	5,258
4	24,453.69	5,419
5	24,720.88	3,824
6	25,135.21	3,522
7	24,076.79	8,617
8	24,367.81	2,966
9	23,009.31	5,595
10	24,469.73	7,360
Avg	24,290.48	5,161
Min	23,009.31	2,966
Max	25,135.21	8,617

Median Count = 90		
Run	**Result**	**Gens**
1	22,477.85	2,640
2	22,092.77	4,303
3	22,010.09	2,523
4	22,535.11	3,462
5	21,510.62	4,910
6	22,024.51	4,563
7	21,889.61	4,099
8	21,369.24	2,725
9	21,271.45	2,939
10	21,860.57	5,451
Avg	21,904.18	3,762
Min	21,271.45	2,523
Max	22,535.11	5,451

Median Count = 100		
Run	**Result**	**Gens**
1	19,838.37	4,410
2	19,383.77	4,093
3	20,329.31	2,949
4	20,674.73	3,173
5	20,751.34	3,004
6	20,210.40	3,040
7	19,966.90	6,392
8	20,150.28	5,823
9	20,122.19	3,896
10	20,200.20	2,598
Avg	20,162.75	3,938
Min	19,383.77	2,598
Max	20,751.34	6,392

Algorithm DAGA-MU; Problem; Set fl1400; Crossover Technique 1

Median Count = 10		
Run	**Result**	**Gens**
1	103,324.29	3,579
2	102,425.47	4,273
3	101,841.67	2,991
4	103,525.47	7,936
5	104,657.19	2,908
6	102,752.87	5,320
7	104,210.93	4,567
8	103,191.44	5,222
9	103,554.74	3,575
10	103,888.05	5,741
Avg	103,337.21	4,611
Min	101,841.67	2,908
Max	104,657.19	7,936

Median Count = 20		
Run	**Result**	**Gens**
1	60,450.53	4,556
2	58,461.03	7,200
3	60,356.07	3,334
4	61,085.19	3,880
5	60,321.62	7,530
6	60,028.90	5,715
7	60,326.36	6,239
8	61,747.33	3,549
9	61,203.48	3,008
10	60,182.91	8,501
Avg	60,416.34	3,008
Min	58,461.03	8,501
Max	61,747.33	5,351

Median Count = 30		
Run	**Result**	**Gens**
1	46,945.14	2,638
2	45,682.05	5,917
3	46,412.97	4,199
4	46,695.50	4,483
5	45,919.72	4,512
6	46,189.53	7,118
7	46,979.75	2,770
8	46,569.13	2,725
9	46,027.46	5,290
10	46,562.92	2,644
Avg	46,398.42	4,230
Min	45,682.05	2,638
Max	46,979.75	7,118

Median Count = 40		
Run	**Result**	**Gens**
1	37,563.28	4,855
2	37,541.83	2,602
3	36,388.83	3,943
4	37,118.59	4,145
5	37,870.62	3,062
6	36,669.11	7,072
7	37,357.47	3,169
8	36,253.23	5,520
9	37,049.90	2,746
10	37,086.13	7,520
Avg	37,089.90	4,463
Min	36,253.23	2,602
Max	37,870.62	7,520

Median Count = 50		
Run	**Result**	**Gens**
1	31,552.07	3,334
2	31,699.40	2,946
3	31,924.53	3,143
4	31,167.75	3,576
5	31,953.02	3,613
6	31,728.72	2,935
7	30,769.12	4,881
8	31,603.40	4,952
9	30,492.70	5,870
10	31,135.56	5,495
Avg	31,402.63	4,075
Min	30,492.70	2,935
Max	31,953.02	5,870

Median Count = 60		
Run	**Result**	**Gens**
1	26,649.38	7,372
2	26,715.01	3,386
3	26,379.66	2,571
4	26,768.11	4,932
5	26,675.90	4,152
6	26,863.64	4,749
7	26,440.20	3,441
8	27,173.60	3,518
9	26,596.05	3,593
10	26,634.56	6,855
Avg	26,689.61	4,457
Min	26,379.66	2,571
Max	27,173.60	7,372

Median Count = 70		
Run	**Result**	**Gens**
1	23,632.85	2,846
2	23,377.32	6,169
3	23,222.47	6,516
4	23,363.25	6,827
5	23,732.98	5,367
6	23,852.30	2,683
7	23,853.53	3,214
8	23,608.91	3,642
9	23,180.84	4,873
10	24,067.28	3,656
Avg	23,589.17	4,579
Min	23,180.84	2,683
Max	24,067.28	6,827

Median Count = 80		
Run	**Result**	**Gens**
1	21,953.28	5,271
2	21,475.42	6,537
3	21,289.82	3,520
4	21,949.51	2,566
5	21,880.16	8,993
6	21,560.90	3,260
7	21,527.88	3,682
8	21,623.40	6,629
9	21,448.33	4,381
10	21,900.68	3,454
Avg	21,660.94	4,829
Min	21,289.82	2,566
Max	21,953.28	8,993

Median Count = 90		
Run	**Result**	**Gens**
1	19,366.66	10,533
2	19,647.26	2,994
3	19,541.80	6,188
4	19,427.60	3,829
5	19,341.48	4,235
6	19,600.76	3,006
7	19,565.46	2,810
8	19,477.26	2,873
9	19,435.33	3,197
10	19,417.17	4,914
Avg	19,482.08	4,458
Min	19,341.48	2,810
Max	19,647.26	10,533

Median Count = 100		
Run	**Result**	**Gens**
1	17,805.57	6,985
2	17,436.57	6,499
3	17,764.73	3,781
4	17,440.81	5,211
5	18,070.48	6,514
6	17,671.57	5,837
7	17,653.62	3,882
8	17,665.16	4,680
9	17,876.84	4,887
10	17,899.39	2,779
Avg	17,728.47	5,106
Min	17,436.57	2,779
Max	18,070.48	6,985

Reference List

Alba, E., & Dominguez, E. (2006). Comparative analysis of modern optimization tools for the p-median problem. *Statistics and Computing, 16*(3), 251-260.

Alp, O., Erkut, E., & Drezner, Z. (2003). An efficient genetic algorithm for the P-Median problem. *Annals of Operations Research, 122*(21-42).

Arthur, D., & Vassilvitskii, S. (2007). *k-means++: The advantages of careful seeding.* Paper presented at the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, Louisiana.

Avella, P., Sassano, A., & Vasil'ev, I. (2007). Computational study of large-scale p-Median problems. *Mathematical Programming, 109(1):89{114, January 2007, 109*(1), 89-114.

Beltran, C., Tadonki, C., & Vial, J. (2006). Solving the *P*-Median Problem with a Semi-Lagrangian Relaxation. *Computational Optimization and Applications, 35*(2), 239-260. doi: 10.1007/s10589-006-6513-6

Bozkaya, B., Zhang, J., & Erkut, E. (2002). An efficient genetic algorithm for the p-median problem. In Z. Drezner & H. Hamacher (Eds.), *Facility Location - Applications and Theory* (pp. 179-205). Berlin: Springer.

Bremermann, H. J. (1962). Optimization through evolution and recombination. In M. C. Yovits, G. T. Jacobi & G. D. Goldstein (Eds.), *Self-Organizing Systems*. New York: Spartan Books.

Chiou, Y., & Lan, L. W. (2001). Genetic clustering algorithms. *European Journal of Operational Research, 135*(2), 413-427.

Correa, E., Steiner, M., Freitas, A., & Carnieri, C. (2001). *A genetic algorithm for the P-median problem.* Paper presented at the 2001 Geneticand Evolutionary Computation Conference, San Francisco, CA.

DeJong, K. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* PhD Doctoral dissertation, University of Michigan, Ann Arbor, Michigan.

Dibble, C., & Densham, P. J. (1993). *Generating intersecting alternatives in GIS and SDSS using genetic algorithms*. Paper presented at the GIS/LIS Symposium, Lincoln, Nebraska.

Estivill-Castro, V., & Torres-Velázquez, R. (1999). Hybrid Genetic Algorithm for Solving the p -Median Problem *Lecture Notes in Computer Science, Simulated Evolution and Learning* (Vol. 1585, pp. 19-25): Springer Berlin / Heidelberg.

Fathali, J. (2006). A genetic algorithm for the p-median problem with pos/neg weights *Applied Mathmatics and Computation, 183*(2), 1071-1083

Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). *Artificial Intelligence through Simulated Evolution*. Hoboken, NJ: Wiley.

García-López, F., Melián-Batista, B., Moreno-Pérez, J. A., & Moreno-Vega, J. M. (2002). The Parallel Variable Neighborhood Search for the *P*-Median Problem. *Journal of Heuristics, 8*(3), 375-388. doi: 10.1023/a:1015013919497

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimizaiton and Machine Learning*. Reading, MA: Addison-Wesley.

Hakimi, S. L. (1964). Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research, 12*(3), 450-459.

Hakimi, S. L. (1965). Optimum Distribution of Switching Centers in a Communication Network and Some Related Graph Theoretic Problems. *Operations Research, 13*(3), 462-475.

Hansen, P., & Mladenovic, N. (1997). Variable neighborhood search for the p-median. *Location Science, 5*(4), 207-226.

Hansen, P., & Mladenovic, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research, 130*(3), 449-467.

Hansen, P., & Mladenovic, N. (2007). Complement to a comparative analysis of heuristics for the p-median problem. *Statistics and Computing, 18*(1), 41-46.

Hansen, P., Mladenović, N., & Perez-Britos, D. (2001). Variable Neighborhood Decomposition Search. *Journal of Heuristics, 7*(4), 335-350. doi: 10.1023/a:1011336210885

Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press.

Hosage, C. M., & Goodchild, M. F. (1986). Discrete space location-allocation solutions from genetic algorithms *Annals of Operations Research, 6*(2), 35-46.

Laszlo, M., & Mukherjee, S. (2006). A genetic algorithm using hyper-quadtrees for low-dimensional k-means clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 28*(4).

Laszlo, M., & Mukherjee, S. (2007). A genetic algorithm that exchanges neighboring centers for k-means clustering. *Pattern Recognition Letters, 28*(16), 2359–2366.

Megiddo, N., & Supowits, K. J. (1984). On the complexity of some common geometric location problems. *SIAM J. Computing, 31*(1), 182-195.

Mladenovi, N., Brimberg, J., Hansen, P., & Moreno-Pérez., J. A. (2007). The p-median problem: A survey of metaheuristic approaches. *European Journal of Operational Research, 179*(3), 927.

Rechenberg, I. (1973). *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien der Biologischen Evolution*. Stuttgart: Frommann-Holzboog.

Reese, J. (2005). Solution methods for the p-median problem: An annotated bibliography. *Networks, 48*(3), 125 - 142.

Reinelt, G. (1991). TSPLIB - A traveling salesman problem library. *ORSA Journal On Computing, 3*(4), 376-384.

Resende, M., & Werneck, R. (2004). A hybrid heuristic for the p-median problem. *Journal of Heuristics, 10*(1), 59-88.

Revelle, C., & Swain, R. (1970). Central facilities location. *Geographical Analysis, 2*, 30-42.

Rolland, E., Schilling, D. A., & Current, J. R. (1996). An efficient tabu search procedure for the *p*-median problem. *European Journal of Operational Research, 96*(2), 329-342.

Samet, H. (2006). *Foundations of Multidimensional and Metric Data Structures*. San Francisco, CA: Morgan Kaufmann.

Tietz, M. B., & Bart, P. (1968). Heuristic methods for estimating the generalized vertex median of a weighted graph. *Operations Research, 16*(5), 955-961.

Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and Computing 5*, 65-85.