An Investigation into Exact Methods for the Continuous $p-$Centre Problem and its Related Problems

A thesis submitted to the University of Kent in the subject of management science for the degree of Doctor of Philosophy

by

Becky Callaghan

2016

Synopsis

This thesis will analyse, investigate and develop new and interesting ideas to optimally solve a location problem called the continuous $p-$centre problem. This problem wishes to locate p facilities in a plane or network of n demand points such that the maximum distance or travel time between each demand point and its closest facility is minimised. Several difficulties are identified which are to be overcome to solve the continuous $p-$centre problem optimally. These relate to producing a finite set of potential facility locations or decreasing the problem size so that less computational time and effort is required. This thesis will examine several schemes that can be applied to this location problem and its related version with the aim to optimally solve large problems that were previously unsolved.

This thesis contains eight chapters. The first three chapters provide an introduction into location problems, with a focus on the $p-$centre problem. Chapter 1 begins with a brief history of location problems, followed by the various classifications and methodologies used to solve them. Chapter 2 provides a review of the methods that have been used to solve the $p-$centre problem, with a focus on the continuous $p-$centre problem. An overview of the location models used in this research is given in Chapter 3, alongside an initial investigative work.

The next two chapters enhance two well-known optimal algorithms for the continuous $p-$centre problem. Chapter 4 develops an interesting exact algorithm that was first proposed over 30 years ago. The original algorithm is reexamined and efficient neighbourhood reductions which are mathematically supported are proposed to improve its overall computational performance. The enhanced algorithm shows a substantial reduction of up to 96% of required computational time compared to the original algorithm, and optimal solutions are found for large data sets that were previously unsolved. Chapter 5 develops a relatively new relaxation-based optimal method. Four mathematically supported enhancements are added to the algorithm to improve its efficiency and its overall computational time. The revised reverse relaxation algorithm yields a vast reduction of up to 87% of computational time required, which is then used

to solve larger data sets where $n \leq 1323$ optimally.

Chapter 6 creates a new relaxation-based matheuristic, called the relaxed p' matheuristic, by combining a well-known heuristic and the optimal method developed in Chapter 5. The unique property of the matheuristic is that it deals with the relaxation of facilities rather than demand points to establish a sub-problem. The matheuristic yields a good, but not necessarily optimal, solution in a reasonable time. To guarantee optimality, the results found from the matheuristic are embedded into the optimal algorithms developed in Chapters 4 and 5.

Chapter 7 adapts the optimal algorithm developed in Chapter 5 to solve two related location problems, namely the $\alpha-$neighbour $p-$centre problem and the conditional $p-$centre problem. The $\alpha-$neighbour $p-$centre problem is investigated and solved where $\alpha = 2$ & 3. A scenario analysis is also conducted for managerial insights by exploring changes in the number of facilities required to cover each demand point. Furthermore, an existing algorithm for the conditional $p-$centre problem is enhanced by incorporating the optimal algorithm proposed in Chapter 5, and it is used to solve large data sets where the number of preexisting facilities is 20. This chapter therefore demonstrates that an algorithm developed in this research can be adapted or used to enhance existing algorithms to optimally solve more practical and challenging related location problems.

Finally, Chapter 8 summarises the findings and main achievements of this research, and outlines any further work that could be worthwhile exploring in the future.

Acknowledgments

I would like to express my gratitude to the people who helped me in this great achievement. I would firstly like to thank EPSRC for their generous funding for this research project, as without them the completion of this Ph.D. would not have been possible.

Secondly, I would like to show my sincerest gratitude to my first supervisor, Professor Saïd Salhi. His brilliant guidance helped me understand and master key skills, and his limitless patience and understanding kept me motivated, especially in times where I lacked confidence. He always had time to talk through ideas, offer insightful suggestions and to give invaluable feedback. I am very thankful for all his hard work.

I would also like to thank my second supervisor, Doctor Gábor Nagy. He was a great support, and always offered alternative viewpoints so that my work could progress in interesting directions. He was also the one who suggested studying for a Ph.D., and showed faith that I could achieve it.

I would like to show my gratitude to my entire family, who showed endless support for my studies. I would especially like to thank my father, Doctor Peter Callaghan, who often took much time to speak about my research with me, understand it and offer interesting observations and advice. I would also like to thank my partner, Kyle Brown, who supported me in many ways through the most challenging part of the research.

I also wish to express my thanks to the Business School at The University of Kent, who helped me complete my Ph.D. by offering support, understanding and kindness towards my disability. I am very grateful and proud to be part of this school in the university.

I would also like to thank God, whom I believe did not only give me the strength and opportunity to do this research, but also supplied me with the support network mentioned above whom I am very grateful for.

Contents

List of Tables

List of Figures

Chapter 1

Introduction to Location Problems

1.1 Introduction

"Set aside for yourself three cities centrally located in the land the Lord your God has given you to possess. Build roads to them and divide them into three parts."

– Deuteronomy 19 v2-3.

Location problems have been around since the dawn of human civilisation. The first recorded location problem is in the Bible, where God commanded Moses to locate three cities of refuge, a city for people who have unintentionally killed, for the people of Israel. Wilamowsky, Epstein & Dickman (1994) proposed an interesting paper that suggested how this problem could have been solved at the time using only the mathematical tools that were available to Joshua. They concluded that the locations Joshua picked in the Bible were indeed the optimal ones.

The aim of this chapter is to clearly outline the research question that will be investigated in this thesis, whilst providing an introduction into location analysis and an overview of the solution methods used. It shall begin by identifying the research question. It will then discuss the history of location analysis before explaining the different classifications for location problems in order to distinguish between the various types. Finally, the chapter will conclude with a section that presents the different methodologies used to solve location problems, with a focus on the exact and heuristic methods used in this research.

1.2 Research Investigation

This research aims to investigate exact methods used to solve a well-known location problem, called the $p-$centre problem, whose aim is to locate p facilities amongst a plane or network of n demand points such that the *maximum distance from a demand point to its closest facility is minimised*. This type of location problem is known as a

centre problem, and it is related to two other well-known types of location problem, namely covering problems and median problems, which will be briefly reviewed later.

Centre problems wish to locate a specific number of facilities amongst a set of demand points, and therefore the maximum distance or time is relaxed in order to establish a solution. Covering problems mirror centre problems by finding the number of facilities required to serve the demand points within a pre-specified distance or travel time. An example of this problem type is the Set Covering Problem, which will be discussed in more detail throughout this research. A median problem wishes to locate a specific number of facilities such that the average distance or travel time between a demand point and its closest facility is minimised. An example of this problem type is the $p-$median problem, see Irawan (2014) for more detail. For clarity, Figure 1.1 shows the relation between these location problems using a tree diagram, and the bold line indicates the research path for this thesis.

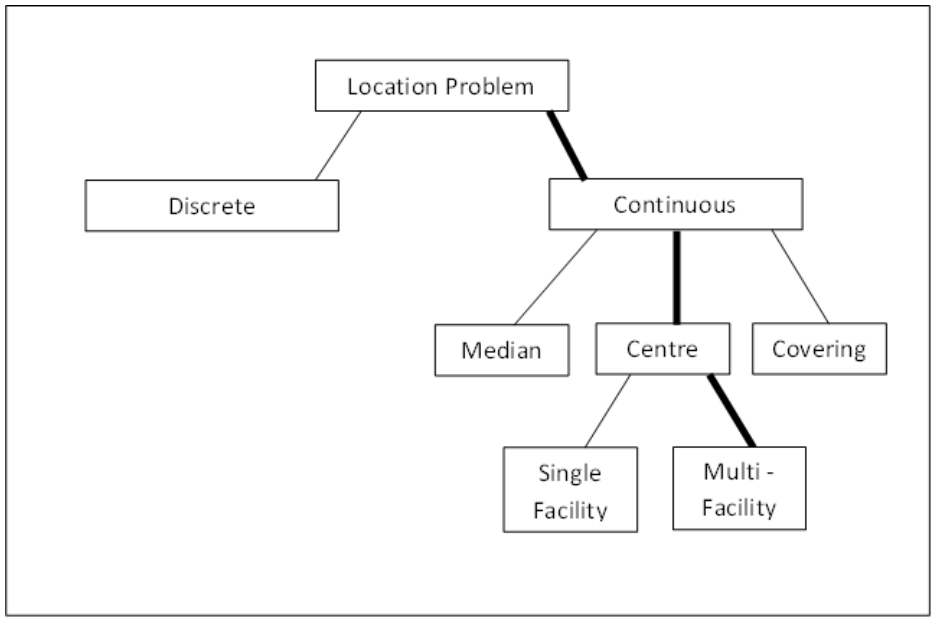


Figure 1.1: Classes of location problems where the bold line indicates our research path

Before analysing this specific location problem in more detail, we shall first provide a brief history into location analysis and discuss the classifications of location problems in further depth.

1.3 A Brief History of Location Analysis with a Focus on Centre Problems

Many authors begin by crediting Pierre de Fermat $(1601-1665)$ for proposing the first basic special median problem. Fermat challenged his pupils by asking the question "Given a set of three points on the plane, find a fourth point such that the sum of the distances to the given three is as small as possible". Fermat and his pupils looked for possible solutions, and most references state that his pupil, Torricelli, proposed the first solution. However, as other mathematicians were working on this problem at the time (such as Cavalieri and Viviani), it is not clear who first proposed the solution to this problem. Therefore, it can be referred to by several names, such as the Fermat Problem or the Fermat-Torricelli Problem.

Torricelli solved the Fermat's problem in the following way. He first joined the three points together to create a triangle. Then, for every pair or points, he drew the equilateral triangle outwards. Three circles are then constructed from the equilateral triangles, and the point where these circles intersect is the median point, or the 'Torricelli point'. This point marked the location of the fourth point. Simpson developed this idea further by constructing three lines which are referred to as the Simpson lines. These lines were made by connecting the outside vertices of the equilateral triangle to the demand point that lies opposite them, see Figure 1.2. The three lines intersect at the Torricelli point.

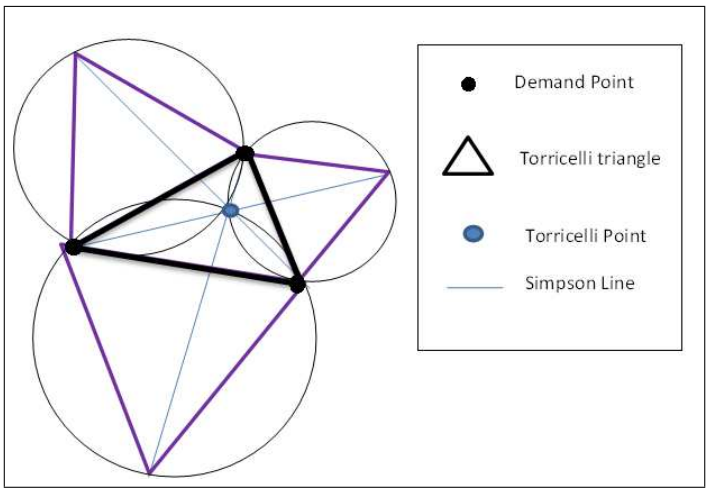


Figure 1.2: Torricelli Points and Simpson Lines

Varignon ($1654 - 1722$) solved a similar location problem proposed by Fermat using a simple mechanical object called a Varignon frame, but this time the number of given points was > 3. First, n holes are drilled in a board, where the board and the holes represented the plane and the location of the given set of points respectively. A piece of string is threaded through each hole, and a weight that is proportional to the respective point's demand is tied to the end. These pieces of string were then all tied to one main piece of string with a knot to keep them bound together. When the string is released, gravity pulled the weights downwards, and the optimal location of the new point is situated at the position of the knot.

The location problem was developed further by an educated landowner called von Thünen. In 1826 he published a book called 'Der Isolierte Straat', that later became one of the founding books on location theory. He developed his ideas around agricultural problems, where he gave a predictive model for rural development. He wanted to allocate crops to land surrounding the town based on transportation and maximising efficiency whilst minimising costs for the local town or city.

In 1857, Sylvester asked the question "Is it possible to find the least circle that can contain a set of given points?" He later answered this question, and today this problem is known as the single facility problem. More detail on the single facility problem can be found in Section 2.2.3, Chapter 2.

Alfred Weber (1909) proposed a location problem that acted as a catalyst for future research into location analysis. Weber wished to find the location of one facility for a set of demand points such that the sum of the weighted Euclidean distances is minimized. This famous location problem is known as the Weber Problem. In the twentieth century, Weiszfeld (1937) developed a simple iterative and convergent procedure that is now commonly used to solve the Weber problem.

Hakimi (1964) published a seminal paper on location problems aimed to locate police stations on a network, such that the number of stations was minimised. He solved the problem using Boolean algebra, and a linear formulation was later presented by

Roth (1969).

Further information on the history of location analysis can be found in Wesolowsky (1993). For an interesting review of location theory, including the variety of problems and techniques used to solve them, see De Berg *et al* (2000), Drezner (2001), and Eiselt & Marianov (2011).

1.4 Classification of Location Problems with a Focus on Centre Problems

Location analysis is part of the field of Operational Research, and has a wide range of mathematical problems that are classified in a variety of ways regarding its specific needs and information. This section will now list and explain the important and relevant classifications for the location problems that will be discussed in this research. For clarity, the choice for each classification type for the location problem studied in this thesis will be highlighted in Table 1.1.

Objective Function

The objective function states what the problem wishes to minimise or maximise. For example, the Set Covering problem wishes to minimise the number of facilities located in order to cover all the demand points. The $p-$centre problem has a slightly different objective function, as it wishes to locate p facilities amongst a set of n demand points, such that the maximum distance from a demand point to its allocated facility is minimised. The $p-$median problem, on the other hand, aims to locate p facilities such that the sum of distances in minimised. Furthermore, the $p-$dispersion problem aims to locate p obnoxious facilities (i.e. power stations or recycling centres) such that the minimum distance from a customer site to an open facility is maximised.

Discrete vs Continuous

If the problem is classified as continuous, then the facilities can be located anywhere in the plane (rather than at pre-sepcified locations). Therefore, there are an infinite number of possible sites for the facilities, and Love *et al* (1988) referred to these problems as "site generating" problems. For the discrete $p-$centre problem, there are specific

nodes on a network or graph that the facilities can be located on. Thus the discrete problem has a finite number of possibilities from which the facilities' locations can be selected.

Capacitated vs Uncapacitated

If a facility is capacitated, this means there is a maximum threshold of service that it can provide. For example, a hospital may be restricted by the maximum number of patients that it can serve, or a warehouse may be limited by its storage size so that it can only serve a limited number of customers. Here, the assignment is achieved by solving the transportation or the assignment problem depending on whether the allocation is single sourced or not.

If the facilities are uncapacitated, then there is no maximum threshold on the amount of service the facility can provide, and hence the assignment of a customer to its open facility is relatively much simpler as each demand point will be allocated to its closest facility.

Single Facilities vs Multiple Facilities

We have previously discussed that centre problems are assigned a given number of facilities to locate. However, as seen in Figure 1.1, the problem can be classified as a single facility problem or a multiple facility problem. If only one facility needs to be located, this becomes the single facility location problem. Many optimal algorithms have been developed to solve this problem, including the Elzinga-Hearn algorithm (EHA) proposed by Elzinga & Hearn (1972) for the $1-$centre problem in the plane and the previously mentioned Weiszfeld's algorithm for the Weber problem. The EHA will be described fully in Chapter 3 as it is used as one of the main ingredients in this research. For the discrete case, the complete enumeration technique (i.e. evaluating all possible sites) is a useful optimal method to solve small-sized problems, and will be revisited in Section 1.5.1.

In practice, several facility location problems wish to locate p facilities where $p > 1$ (e.g. the $p-$centre problem or the $p-$median problem). As this generates a larger, more complex problem, multiple facility location problems take more computational effort

to solve. This therefore means that some location problems cannot be solved optimally.

Furthermore, in other cases, p can also be a decision variable such as the uncapacitated/ capacitated location problem where a fixed cost of establishing a facility is required. Note that the former case can be used for scenario analysis by varying the value of p.

Conditional vs Unconditional

A location problem is classified as conditional if p facilities need to be located, given that q facilities already exist. For example, if we wish to locate three hospitals in a county that already has five, we would want to take the location of the existing five hospitals into consideration before locating the new ones. If $q = 0$ (i.e. there are no existing facilities to consider), the problem reduces to the unconditional case, see Berman and Drezner (2008) for further details. This variant, with respect to the $p-$centre problem, will be studied in Chapter 7.

Constrained vs Unconstrained

A location problem is said to be constrained if there are areas where facilities cannot be located. For example, it is often undesirable to locate a facility somewhere heavily populated, and it is impossible to locate a facility on a lake. Furthermore, the problem may wish to locate the facilities, such as mountain rescue centres, in specific areas of the plane or network. The areas in the plane or network where the facilities cannot be situated are called forbidden regions, see Plastria (2002) and Suzuki & Okabe (1995). If there are no forbidden areas, then the problem is classified as unconstrained.

Deterministic vs Probabilistic

This type of classification determines the uncertainty of a model regarding its predicted behaviour in the future. This depends on whether the underlying network of the model is static or dynamic. If it is static, then elements of the model, such as the position of the demand points, the travel time or the distance from the demand points to the facilities, do not change over time. In other words, the solution to the current location problem will still be relevant and correct in the future. This location problem is classified as deterministic.

However, if the model has elements of uncertainty following certain density functions for the future, then it is classified as probabilistic. For example, say we wish to locate p fire stations on a set of n demand points. An element of uncertainty could be the travel time from the fire station to a demand point as this may change in the future due to factors like traffic cycles, volume of traffic, the population or a change in the road network. In this class, some problems are more sensitive than others. For example, the $p-$centre problem can be more affected by a change of one new customer (e.g. the new customer happens to be further away) than the $p-$median problem (e.g. the effect of a new customer on the average sum may not be so great) due to their corresponding objective functions (i.e. MinMax vs. MinSum). Investigations for this problem type for the single facility problem include Wesolowsky (1977) and Berman *et al* (2003) who study the weighted single facility location problem. Furthermore, Averbakh & Berman (1997) studied the weighted $p-$centre problem where the weights of each demand point were uncertain based on probability density functions.

Demand Points

Demand points can be situated either a) anywhere on the plane or graph or b) in specific areas. For example, if the $p-$centre problem wishes to locate p facilities amongst a network of demand points, the demand points may only be able to be situated at the nodes of the network. Moreover, the demand can be based on coverage areas rather than specific points, such as in agriculture and crime, see Murray, O'Kelly & Church (2006).

Oriented vs Non-Oriented

Establishing whether a location problem is oriented or non-oriented (or bi-directional) is extremely important. This could easily make the problem asymmetric, which is likely to violate the triangle inequality and so rendering the problem even harder to solve.

Distance Metrics

As this report will focus on minimising the maximum distance, it is important to know how distance is measured and defined in this research. Drezner & Hamacher state the five most commonly used distances in location analysis, namely the Euclidean distance,

rectangular distance, max distance, hexagonal distance and Minkowski distance, see Drezner & Hamacher (2001) for further details. The most familiar measure, which is used in our research, is the Euclidean distance representing the straight line distance between any two points.

Classification	**Our Research**
Objective Function	Minimise maximum distance
Discrete vs. Continuous	Continuous
Capacitated vs. Uncapacitated	Uncapacitated
Single Facilities vs. Multiple Facilities	Multiple Facilities
Conditional vs. Unconditional	Unconditional & Conditional
Constrained vs. Unconstrained	Unconstrained
Deterministic vs. Probabilistic	Deterministic
Demand points	Anywhere in the plane
Oriented vs. Non-Oriented	Non-Oriented
Distance	Euclidean

Table 1.1: The classification of the $p-$centre problem investigated in this research

The relevant, interesting and common classification types for location problems have been discussed, allowing us to highlight in Table 1.1 the classification option for the location problem investigated in this research.

1.5 An Overview of Methodologies used for Location Problems

The two main methodologies used to solve location problems are exact and heuristic methods. Exact methods guarantee the optimal solution if they are run until the end, whereas heuristics give an approximate solution in relatively short time. Exact methods are therefore the most desirable to utilise if possible, and they are often used to solve easier combinatorial or global optimisation problems or NP-hard location problems that are relatively small. However, as NP-hard problems increase in size, they often become difficult or impossible to solve using exact methods only. This may be due to the amount of computational time and effort they require, or practical issues such as a lack of computer memory. Therefore, heuristic methods may be incorporated so that a good, but not necessarily optimal, solution may be found in a reasonable amount of computational time. Moreover, heuristics have the added advantage of be-

ing used for scenario analysis due to their speed.

This dissertation focuses on solving the continuous $p-$centre problem using exact methods, and so this section will discuss the relevant exact methodologies that were put into practice during this research. However, heuristic methods were also incorporated into several algorithms in order to obtain the optimal solution more efficiently. Therefore, this section will also analyse several heuristic methods that have some relevance to this research.

1.5.1 Exact Methods

The two main exact methods used to solve location problems are complete enumeration and integer linear programming. Note that dynamic programming is also well-known but less used in the field of location problems, especially in the research we are investigating here.

Complete Enumeration

Complete enumeration optimally solves combinatorial problems by evaluating every combination and then choosing the best solution. For example, the discrete $p-$centre problem wishes to locate p facilities from a total of m potential facility locations. The total number of combinations for the facilities is therefore $C_p^m = \frac{m!}{p!(m-p)!}$. For example, if we wish to locate 2 facilities given there are 10 potential facility locations, there will be a total of 45 combinations to check. If the number of potential facility locations were to only increase by 5 (i.e. $m = 15$), the total number of combinations to check increases to 105. It is clear to see that as the problem size increases, this method becomes increasingly difficult, or even impossible, to use as the number of combinations becomes too great. There are however some interesting reduction techniques that are developed to eliminate irrelevant combinations using memory of previous calculations, see Alharbi (2010). Though larger problems were optimally solved, the handicap of this blind search remains valid for larger problems (e.g., $n = 100$ and $p = 5$). It is worth noting that this simple optimal method can be practical and easy to implement if the problem happens to be small enough, and hence there is no need for expensive and sophisticated commercial solvers.

Integer Linear Programming

Integer linear programming is a mathematical tool used to optimally solve a variety of combinatorial problems, such as location problems. This optimal method can be used to solve large location problems that cannot be solved using other methods such as complete enumeration with the aid of powerful commercial optimisation solvers such as IBM ILOG CPLEX, LINDO and XPRESS amongst others.

In an integer linear programming model, all the variables are of integer values. This includes decision variables, which can specifically binary (e.g. open/closed, yes/no, off/on) or an integer (e.g. the number of facilities located). Many location problems can be formulated as integer linear programming models including the relevant formulations of location problems studied in this research, see Chapter 3.

Among the techniques used include the Branch & Bound Method, the Cutting Plane Method and the Branch & Cut Method.

The Branch & Bound Method (BBM) is the most well known method used to solve integer linear programming problems, and it consists of two steps. Let us wish to solve the continuous $p-$centre problem using the BBM, where an upper bound on the optimal solution has been obtained. In this instance, the objective function wishes to minimise the solution value. The first step (i.e. branching) partitions the set of feasible solutions into subsets. The LP solution value for each subset is then obtained, thus yielding a set of lower bounds (i.e. bounding). If a subset's lower bound exceeds the upper bound, this subset can be excluded, or fathomed, from the search. The remaining subsets are partitioned further and further until a feasible solution is found such that the solution value is the smallest lower bound found amongst all the subsets. The solution is optimal since there is no other subset yielding an improved solution. For more information, see Lowe (2011).

The Cutting Plane Method (CPM), first introduced by Gomory (1958), solves the integer linear programming problem by first solving the linear programming problem. If an integer solution is found, then the process stops. Otherwise, a new constraint,

known as a Gomory cut, is added to the problem that reduces the feasible region and therefore 'cuts' off the non-integer solution. The problem is then solved again with the new constraint usually using the dual simplex method. Each time a non-integer solution value is obtained, the feasible region is reduced further by adding another constraint and so allowing the problem to converge towards the optimal solution.

The CPM was found to take a large amount of computational time and effort as the constant addition of new constraints created a more complex problem. The weakness of CPM is that no integer solution can be found until the optimal final solution is obtained, whereas the BBM has an advantage of producing intermediate integer solutions. An effective combination of the BBM and the CPM, called the Branch & Cut method, was developed and found to be very efficient. It is used in many powerful commercial optimisation software packages, such as IBM ILOG CPLEX, LINDO and XPRESS amongst others.

1.5.2 Heuristic Methods

The word 'heuristic' originates from the Greek word that means discover or explore in the wider sense. For location analysis, heuristic methods have been developed so that concrete solutions may be obtained for large problems that cannot be solved using optimal techniques. Their efficiency is examined by two main criteria, namely the quality of the solution value found and the amount of computational time required to obtain the solution. Therefore, the most powerful heuristics are the ones that find a high quality solution in a reasonable amount of computational time. These heuristics have been used in many academic areas of research such as business, economics, medicine and engineering; see Salhi (2006) for an overview of heuristic search.

Improvement-Based Heuristics

An improvement-based heuristic only accepts improving solutions at each iteration. An example is Cooper's locate-allocate heuristic, which is discussed later in Section 2.2.2, Chapter 2. The heuristic starts with an initial solution (i.e. p facilities) and begins a cycle of locating all demand points to their closest facility to generate a new set of p facilities until the solution does not change (i.e. the heuristic has become caught in a local minimum). Although these type of heuristics are useful, their weakness is

becoming trapped at local minima. Therefore, they are most effective to use when there is either only one local minimum/maximum (i.e. a global one), or if they are incorporated as an extra method, amongst another heuristic or optimal method, to improve the solution value further. Note that the first scenario happens when the objective function is convex (concave) and hence there is only one minimum (maximum). The continuous $p-$centre problem is non-convex, and so the optimal solution cannot be guaranteed by Cooper's locate-allocate approach.

Furthermore, other methods, namely local search or improvement/refinement based methods, can be applied to help improvement-based heuristics, such as Cooper's locate-allocate heuristic, explore further areas of the solution space. An example is the local search method, which changes the current feasible solution in an intelligent way so that an improved solution may be found, see Salhi (1997) for further details. In this research, a heuristic developed by Drezner (1984a), called the H_2 heuristic, pairs a locate-allocate heuristic with a swap-based local search in order to find an improved feasible solution for the continuous $p-$centre problem. As H_2 is also used in this research, it will be revisited in Section 2.2.2, Chapter 2.

Metaheuristics Used To Solve the $p-$centre Problem

Several powerful heuristics, known as metaheuristics or modern heuristics, have been adapted or enhanced to solve the discrete and continuous $p-$centre problem. This section will discuss several of these interesting and relevant heuristics.

An important heuristic for the $p-$centre problem, developed by Brimberg & Mladenović (1996), is the Variable Neighbourhood Search (VNS) heuristic which was designed to avoid local optimality in order to obtain a tight feasible solution for discrete and continuous location problems. This metaheuristic systematically changes the neighbourhoods to see if a better solution can be found, see Mladenović & Hansen (1997). Results using this heuristic is used in this thesis for comparison purposes when analysing the solution quality in Chapter 6. Furthermore, a detailed description of this heuristic is given in Section 2.2.2, Chapter 2.

Davidović *et al* (2011) proposed a metaheuristic method called Bee Colony Optimiza-

tion (BCO). Bees find the best places for food by searching an area individually and announcing the quality of the area they found to the whole hive. The better the quality of the area, the more likely other bees will start to find food in that area. The BCO metaheuristic was formed of two stages, namely the forward pass and the backward pass. In the forward pass, agents (i.e. the artificial bees) are sent out to explore the solution space and find a feasible solution. During the backward pass, the agents come back and announce what they have found (i.e. state the solution value of the feasible solution). The better the solution value, the more likely other agents will not stay loyal to their solution and will have to pick from the other advertised objective functions values. Davidović *et al* found that the BCO was effective at finding solutions for the discrete $p-$centre problem for small problems. However, as the problem sizes increased, it was found to be less effective. This motivated the development on an improved method, namely the BCOi, that modified the generated solutions in order to establish an improved one, rather than 'starting from scratch' or adding a new component to a partial solution. The modification consisted of allowing $p+q$ facilities to serve the n demand points (i.e. creating an infeasible solution in order to reduce the solution value) and then removing q facilities in a greedy manner. The BCOi yielded high quality solutions with a negligible increase in computational time. Moreover, the idea of perturbing the solution by adding q facilities and then removing them is a simpler version of the perturbation method originally developed by Salhi (1997), and very recently extended for the continuous $p-$centre problem by Elshaikh *et al* (2016).

Another interesting heuristic used to solve the discrete $p-$centre problem is the Harmony Search heuristic, which was inspired by the way a musician adjusts musical notes according to the previously played ones to create a melodic tune. The heuristic performs in a similar way by selecting a number of initial solutions and creating a new 'harmony'. Kaveh & Nasr (2011) modified a harmony search in order to solve the conditional and unconditional discrete $p-$centre problem. Firstly, they developed the algorithm so that the diversification and intensification of the solution can be controlled. Secondly, a new facility (or 'note') is selected based on its location and distance to the current facility. Thirdly, a greedy heuristic is incorporated into the model so that each new harmony is enhanced such that it is more likely to yield an improved solution.

Rabie, El-Khodary & Tharwat (2013) used a metaheuristic called the Particle Swarm Optimization (PSO), developed by Kennedy & Eberhart (1995), to solve the continuous $p-$centre problem. The PSO is a population-based metaheuristic, and it is inspired by the way birds fly in union together in close proximity but never touch. As the solution space in Rabie *et al*'s study was continuous, the main modifications they made were to "generate the swarm on space/plane limits instead of generating it on arcs of networks". They concluded that the developed algorithm was not only simple and easy to apply, but also efficient.

1.5.3 Integrating Exact and Heuristic Methods

As previously mentioned, the aim of this research is to investigate new and improved methods to optimally solve the continuous $p-$centre problem. As heuristic methods cannot guarantee optimality, one may think that they have little relevance in this research. However, heuristic methods are highly important when optimally solving a problem as a good feasible solution can be embedded into an optimal method so that it does not need to 'start from scratch'. This means that there is a direct relation between the quality of the heuristic solution, and whether or not the optimal technique will obtain optimality within the reasonable amount of computational time. The most common way of integrating the two methods is obtaining a good quality solution value (i.e. reasonably close to the optimal solution) using a powerful heuristic that can be used in optimal methods. For example, in our research, the results of the H_2 heuristic, Cooper's multi-start heuristic and the VNS heuristic are set as the initial upper bounds for the optimal methods used.

A hybrid of exact and heuristic methods is also known as a matheuristic, where at least one aspect contains an optimal method. In Chapter 6, a new matheuristic, inspired by the hybridisation of some heuristics and exact methods previously discussed, is created to find a tight upper bound or a good feasible solution that can be used to optimally solve the continuous $p-$centre problem.

1.6 Summary

This chapter introduced the research question that will be throughly investigated and answered in this thesis. It then proceeded to give an introduction into location analysis by first discussing briefly the history of location analysis with a focus on centre problems. The different classification types of location problems were then explained, and the classification options for the specific problem that this research focuses on has been highlighted for clarity. Finally, the different types of exact and heuristic methods used to solve this location problem were discussed, with a focus on the optimal and approximate methods used in this research.

The next chapter provides an extensive literature review of the recent and relevant studies in the area of location analysis with a focus on centre problems and the continuous $p-$centre problem in particular.

Chapter 2

A Literature Review of Location Problems with a Focus on the Continuous $P-$Centre Problem

2.1 Introduction

The purpose of this chapter is to provide an account of the research that has been investigated into both the continuous $p-$centre problem and its related location problems. The chapter is divided into two main sections. The first section deals with the literature that was directly used and applied to this research. This includes exact methods that were utilized or enhanced, the heuristic methods that were incorporated into the optimal algorithms, and the related location problems that were examined in this research, such as the $\alpha-$neighbour $p-$centre problem and the conditional $p-$centre problem. The second section explores those methods that, although not directly used in this research, show the developments, enhancements and adaptations in order to provide some lead ideas that were taken into account in this study. These techniques are designed for the discrete $p-$centre problem, the maximal coverage problem and other related location problems.

2.2 Literature Investigated & Applied in this Research

In this section, the literature that was directly investigated and used in this study will be described. This includes a) the heuristic methods that were incorporated into our research, b) the exact algorithms that were developed and enhanced to create a more efficient algorithm for the continuous $p-$centre problem and c) the related location problems.

2.2.1 Introduction

The $p-$centre problem is an old but interesting topic of research in location analysis. It was first mentioned by Miehle (1958), who proposed three main methods to minimise link lengths in networks. One of the methods is referred to as the 'soap film method'. When air is blown into soap, the soap film expands to create a sphere that we know as 'bubbles'. Miehle observed that the size of the bubble is minimised such that it can still hold the maximum potential energy carried inside, and thus the basic ideas for the $p-$centre problem began. The problem was then later formulated by Cooper (1963), and Hakimi (1964) was the first to find the discrete and continuous centre in a network of demand points.

Megiddo & Supowit (1984) showed that the $p-$centre problem is NP-hard for variable p. In other words, the larger the problem size, the harder it becomes to obtain the optimal solution (i.e. the computational effort grows exponentially as the problem size gets larger) leading to a problem that cannot be solved in polynomial time. Even the most powerful and modern computers experience difficulties solving large location problems due to the lack of computer memory or the requirement of an exceptionally high amount of computational effort. Therefore, to save on time and expense, it has been desirable to use alternative methods, such as heuristic and metaheuristic algorithms, so that approximate solutions may be obtained within a reasonable amount of time. However, due to large advancements in technology and computer power, it has now become a realistic possibility that optimal solutions for large problems can now be pursued. Furthermore, the combination of exact and heuristic methods provide fast and powerful algorithms that allow large, complex problems to be solved optimally in a reasonable amount of computational time.

2.2.2 Heuristic Methods

Cooper (1963) was the first to formulate the general location problem. He proposed an exact equation for locating p facilities anywhere in the plane of n demand points using differentiation. However, these problems could only be solved in a reasonable amount of time where $n \leq 10$. Therefore, he developed a second method where all the possible combinations of p temporary facilities from the n demand points are obtained,

and the demand points are allocated to their closest temporary facility. The configuration of facility locations that yielded the minimum distance or cost are taken as the final locations; thus attaining the optimal solution for the discrete case. This feasible solution for the continuous problem was then incorporated into the optimal methods to determine the exact solution for the continuous $p-$centre problem. This process of embedding the heuristic solution into the optimal methods allowed larger problems to be solved optimally. However, this method would not be computationally attractive for very large industrial problems and so Cooper suggested that the development of other heuristics may be worthwhile.

Cooper (1964) later introduced four heuristics for the continuous $p-$centre problem. The first he named the Destination Subset Algorithm, which is described above, and the second heuristic was named the Random Destination Algorithm. This is similar to the Destination Subset Algorithm, with a difference of selecting the initial facility locations at random rather than obtaining every combination. The third heuristic was called the Successive Approximations Algorithm, which involved adding new facilities to the solution space gradually until there are p facilities. The final heuristic was called the Alternate Location and Allocation Algorithm, which consisted of dividing the group of demand points into p subsets, and then finding the optimal solution of the $1-$centre problem for each subset using an established exact method such as the Elzinga-Hearn algorithm that will be described in Section 2.2.3. This research pairs the Random Destination Algorithm with the Alternate Location and Allocation Algorithm to form Cooper's multi-start heuristic, or Cooper's locate-allocate method, which is investigated in Chapter 3.

A review of location-allocation systems, including variations such as capacitated models, can also be found in Scott (1970).

Drezner (1984a) proposed two very interesting and effective heuristics. The first heuristic, H_1, is much like Cooper's locate-allocate method mentioned previously. Drezner's heuristic begins with an initial p locations for the facilities, and the demand points are allocated to their closest facility forming p clusters of demand points. The plane is then partitioned into p polygon areas such that each polygon encompasses one facility and

the area in the plane that lay closest to that facility. The union of these polygons is known as a Voronoi diagram, which will be revisited in Section 6.2.2, Chapter 6. The centres of these p polygons become the new p facility locations, and the process repeats until the configuration of the facilities do not change. In order to improve the heuristic solution and avoid becoming trapped in a local minimum, Drezner suggests pairing the H_1 heuristic with a perturbation method. This step consists of re-allocating demand points to another facility in an attempt to find an improved solution. He improves this method by demonstrating that the only demand points that need to be considered for re-allocation are the critical points of the solution circle (i.e. the demand points forming the largest circle in the solution). This is because the solution value cannot improve unless the size of the solution circle is decreased. Therefore, different arrangements of these critical points are tested until an improved solution is found. Drezner suggests alternating this perturbation method with the H_1 heuristic to obtain an improved feasible solution. This union of the two methods will be referred to as the H_2 heuristic, and it is used in Chapter 4 to find initial upper bounds for the solution value of the $p-$centre problem.

Mladdenović, Labbé & Hansen (2003) adapted the $VNS-$based heuristic so that it could be used to solve the $p-$centre problem. Neighbourhoods based on moving facilities, as well as re-allocating customers, are explored with interesting results. Elshaikh, Salhi & Nagy (2015) used VNS to investigate the $p-$centre problem in the plane. Firstly, they incorporated learning into the search which meant useful parameters were identified and controlled. They also developed the local search step using two enhancements. Firstly, they allocated a critical point from the largest circle to another facility in an attempt to decrease the size of the largest covering circle. This idea uses the observation noted by Drezner stating that the solution value cannot improve unless the radius of the largest covering circle decreases. It is important to note that this reasoning is also considered when developing the matheuristic in Chapter 6. Secondly, the authors removed 'non$-$promising' facilities. That is, the circles that only serve the critical points and so do not encompass any other demand points. The enhanced VNS algorithm gave promising results as it was easy to implement, and outperformed other known heuristics for TSP-Library data sets such as $rat575$, $rat783$, $pr1002$ and $rl1323$. The results of this enhanced VNS are used as an initial upper bound for the optimal

algorithms developed in Chapters 4, 5 and 6.

More recently, Elshaikh *et al* (2016) developed a powerful perturbation heuristic to solve large continuous $p-$centre problems, see Section 2.3.2 for further details.

2.2.3 The Elzinga-Hearn Algorithm (The Single Facility Problem)

The single facility location problem, or $1-$centre problem, wishes to locate one facility on the plane amongst a network of discrete or fixed demand points, such that the maximum distance from the facility to any demand point is minimised. As mentioned previously, Sylvester (1814-1897) first proposed the problem of finding the smallest circle that encompassed a set of demand points and later solved it. More details can be found in Wesolosky (1993) and references therein. As the method used here is the basis for the multi-facility problem, a brief review on this issue is given next.

One of the most common methods to solve this problem to optimality is the geometry-based algorithm proposed by Elzinga & Hearn (1972), referred to as EHA. The cornerstone to this method is based on the theorem that the optimal location for the $1-$centre problem is equidistant from two or three demand points. The EHA begins by choosing two demand points at random to form the diameter of the circle. The algorithm then checks if all the demand points are covered by the circle or not. If so, then the minimum covering circle has been found. Else, the covering circle is enlarged by reforming it from one or two covered point/s and an uncovered demand point. This process continues until all demand points are encompassed by the covering circle.

Drezner & Shelah (1987) state that the EHA was found to be very efficient when tested on randomly generated problems. There is no cycling as a larger (or same size) circle is found at each iteration, and there are a finite number of steps as the EHA always converges towards the optimal solution. These two properties create a simple, deterministic and a commonly used algorithm to optimally solve the $1-$centre problem. A detailed description of the algorithm can be found in Section 3.3.1, Chapter 3 as it was incorporated into the algorithm developed in Chapter 4.

Recently, Elshaikh, Salhi & Nagy (2015) enhanced and modified the EHA to solve the unweighted and weighted $1-$centre problem. The first enhancement consisted of a better selection method for the initial points. The four points that had the minimum and maximum x and y coordinate values were found, and from these four extreme points, the two points that had the greatest weighted distance were then chosen as the two initial points. This improved selection method decreased the required amount of the computational time with an average decrease of over 46%. Furthermore, they also proposed a second enhancement for the selection of the uncovered point. Instead of choosing a point at random, the one with the greatest weighted distance from the current circle's centre was selected. This enhancement further reduced the computational time greatly with an average decrease of over 42%. These two enhancements to the EHA are both important and relevant to this research as such ideas were incorporated into the optimal algorithm developed in Chapter 4.

2.2.4 Optimal Methods for the $p-$centre Problem

There are two optimal type approaches used to solve the continuous $p-$centre problem in this study, namely the maximal circles-based method and the relaxation method.

(i) The Maximal Circles-Based Method

Drezner (1984a) proposed an interesting exact method to find the optimal solution value to the continuous $p-$centre problem using a subset of potential facility covering circles called *maximal circles*. The process begins with an initial upper bound, Z, which was found using an established heuristic method, such as Cooper's locate-allocate method or the H_2 heuristic proposed by Drezner (both previously described in Section 2.2.2). A circle is defined as maximal based on the upper bound Z. The set of maximal circles is treated as the set of potential facility locations, as Drezner proves that the largest covering circle in the optimal solution is a maximal circle. By narrowing the size of the set of potential facility locations, the problem size is efficiently decreased such that larger problems can be solved optimally. Drezner's algorithm consists of a) finding the set of maximal circles and b) searching for a better solution (i.e. a feasible solution with a solution value smaller than Z) using this set. If a better solution is found, the upper bound Z is updated as the new, improved solution value, a new set of maximal circles is attained and the process begins again. If a better solution value cannot be

found with the set of maximal circles, the optimal solution is the current upper bound Z. A detailed description of Drezner's algorithm can be found in Figure 4.1, Chapter 4.

This method shows much potential as it yields a relatively small problem size for large problems, and so it may be beneficial to be investigated further. Drezner's method is examined thoroughly in Chapter 4.

(ii) The Relaxation-Based Method

Relaxation is a simple method used to solve large problems by breaking them down into smaller sub-problems and successively solving them. The classic algorithm, first suggested by Handler and Mirchandani (1979), begins with an upper bound of infinity, and either updates the upper bound or adds demand points to the subset at each iteration until optimality is reached. Alternative variations to the classic relaxation algorithm have been researched and will also be described in this section. This method has great importance to this research, as it forms the backbone to the research given in Chapters 5, 6 and 7.

Chen & Handler (1987) were one of the first to propose a relaxation-based algorithm that solved the continuous $p-$centre problem. First, they explained that a finite set of potential facility locations (from the infinite set) could be attained by finding all the *critical circles*, where a critical circle has either a) three or more demand points on its circumference, b) two demand points forming the ends of the diameter or c) a null circle consisting of the single demand point (these points shall be referred to as *critical points* in this study). Therefore, the full set of potential facility locations can be calculated as $\binom{n}{3} + \binom{n}{2} + n$, where $\binom{n}{3}$ refers to the number of circles created from three demand points, $\binom{n}{2}$ refers to the number of circles created from two demand points and n is the number of demand points. Furthermore, the number of critical circles can be decreased further when analysing the geometry of the demand points forming them. For example, all circles made from three points that form an obtuse or right$-$angled triangle can be discarded, as the circle created from the two points furthest from each other would incorporate all three points. This method of finding all critical circles formed from three demand points shall be referred to as the 'angle method'. Chapter 3 proposes a new method that finds these critical circles more efficiently and in less

computational time.

The foundation of Chen & Handler's relaxation-based algorithm is built on the well-known theorem stating that among all the optimal solutions to the $p-$centre problem, at least one of them consists of p critical circles encompassing all the demand points. Therefore, by finding the full set of possible facility locations, a finite set of potential facility locations has been obtained which allows us to solve the problem optimally. Their method also incorporated the observation that the solution for the $(p-1)-$centre solution yielded an upper bound for the $p-$centre solution. This allowed tight upper bounds to be obtained, and so meant many circles can be discarded if their radius size exceeded the upper bound. Therefore, this reduced the number of calculations and allowed the problem to require less memory and relatively less computational time to be solved. The authors gave a small example where $n = 10$ and $p = 1, 2, \ldots, 10$. Their method starts by solving the $1-$centre problem using established methods on a very small subset of demand points. An arbitrary point was then added and all the possible critical circles constructed from the subset of demand points are found. A solution is obtained for the relaxed problem (i.e. the subset of demand points), and feasibility for the full problem is checked. If the solution for the sub-problem is feasible for the full problem, another arbitrary point is added to the subset and the process continues until p facilities have been located. If not, the point farthest from its closest circle centre is added to the subset of demand points and another solution for the same number of facilities is found.

Chen & Chen (2009) proposed two new and interesting relaxation algorithms based on the classic relaxation algorithm to solve the discrete and the continuous $p-$centre problem optimally. The classic relaxation algorithm optimally solves large problems by breaking them down into smaller sub-problems that are easier to solve. Every time a feasible solution with a value less than the current upper bound is obtained for the sub-problem then, much like Chen & Handler's (1987) approach, feasibility for the full problem is checked. If it is feasible for the full problem, then the upper bound is updated. Else, another demand point is added to the subset. If a feasible solution with a value less than the current upper bound cannot be found for the subset, then the current upper bound is the optimal solution value. The authors presented two

improvements for the classic relaxation method. The first improvement updates the upper bound more efficiently by treating every feasible solution for the relaxed problem as a feasible solution for the full problem. The second improvement adds more than one point to the relaxed subset at a time to reduce the number of 'uninformative iterations' and therefore creates a more efficient algorithm. However, the number of demand points that are added to the subset needs to be carefully balanced between decreasing the number of iterations and keeping the number of circles created to a minimum. If too many demand points are added, this creates a large problem to solve and negates the use of the relaxation method. Chen & Chen state that this is an area that could be researched in more depth, and this gap of knowledge is investigated in Chapter 5.

Chen & Chen (2009) also suggested two new relaxation algorithms, namely the reverse relaxation algorithm and the binary relaxation algorithm. In short, the reverse relaxation algorithm starts with a lower bound of 0, and constantly updates it until optimality is reached. The binary relaxation algorithm has "the best of both worlds" as it updates either the lower or upper bound at each step. The algorithm starts with an initial coverage distance $\frac{LB+UB}{2}$, where LB denotes the lower bound and UB denotes the upper bound. The LB and UB values are updated throughout the algorithm depending on whether a solution can be found within the coverage distance for the subproblem or for the full problem. A detailed description of these two algorithms can be found in Figures 5.2 & 5.3, Chapter 5, as both are tested in order to determine which one shows the most potential for further improvement. The results show justification for choosing the reverse relaxation for further development.

2.2.5 Location Problems Related to the $p-$centre Problem

The Set Covering Problem

The set covering problem (SCP) is one of the most well-known location problems. The aim is to find the minimum number of facilities in a plane or network required such that all demand points are covered. Hakimi (1964) first used the SCP to solve the problem of locating the minimum number of police stations amongst a set of demand points. The problem was later formulated by Toregas *et al* (1971), who stated that the simplicity of the SCP meant that larger problems could now be solved for the first

time. For more information on the SCP, see Schilling *et al* (1993).

The SCP is revisited in Section 3.2.1, Chapter 3, where the full formulation is given, and it is directly used in this research by being embedded into the optimal algorithm proposed in Chapter 5.

A relevant variation of the SCP is the multi-level set covering problem that was first defined by Toregas (1971). This is where each demand point is covered by a specific number facilities as opposed to only one facility. The coverage need may differ between the demand points (e.g. some may need to be covered by one facility whereas other may need to be covered by three facilities). This creates a more complex problem as infeasibility becomes an issue if co-location of the facilities (i.e. facilities situated in the same place) is not allowed. Church & Gerrard (2003) investigated the multi-level set covering problem where co-location was not allowed. Furthermore, they also investigated the instance where the number of facilities needed to cover a demand point varied. This idea is incorporated into Section 7.2.4, Chapter 7, where a scenario analysis is performed on a relaxation-based algorithm. Other research in this area includes the development of algorithms where co-location of facilities is accepted, such as Hogan & Revelle's (1986). This idea is also incorporated into the algorithm developed in Section 7.2, Chapter 7. Finally, the research for the multi-level set covering problem provides the backbone for a variation of the $p-$centre problem, namely the $\alpha-$neighbour $p-$centre problem, which will be discussed in more detail in Section 2.2.5.

The $\alpha-$neighbour $p-$centre Problem

In the classic $p-$centre problem, the maximum distance between a demand point and its closest facility is minimised. This idea is expanded for the $\alpha-$neighbour $p-$centre problem, where the maximum distance between a demand points and its closest α facilities are minimised. Therefore, the objective is to minimise the maximum distance between the demand point and its α^{th} furthest facility, which is equivalent to ensuring that each demand point is covered by at least α circles. This means if a demand point's $\alpha - 1$ closest service facilities fail, it is still covered by a facility. Therefore, the solution to this problem provides extra security for location problems where the loss of facilities may be realistic, such as the failure of a power station or the closure of a hospital.

The $\alpha-$neighbour $p-$centre problem was first presented by Krumke (1995), who provided an approximation algorithm to solve the $\alpha-$neighbour $p-$centre problem on a network with an approximation factor of 4 when $\alpha \geq 2$. Khuller *et al* (2000) improved this by producing an approximation algorithm that had an approximation factor of 3 for $\alpha > 2$, and 2 for $\alpha = 2$.

Chen & Chen (2013) devised an optimal algorithm for the continuous $\alpha-$neighbour $p-$centre problem by adapting two well-known optimal algorithms. These two algorithms were an adjustment of Minieka's (1970) algorithm (see Section 2.3.4) used to solve the discrete and continuous $p-$centre problem, and the classic relaxation algorithm, previously discussed in Section 2.2.4. Experimental results revealed that the adapted classic relaxation algorithm was more efficient at optimally solving the $\alpha-$neighbour $p-$centre problem than Minieka's adapted algorithm, and this comparison became clearer as the size of the data sets increased (i.e. comparing $n = 48, 101$ & 150). The computational time required to optimally solve the $\alpha-$neighbour $p-$centre problem using the adapted classic relaxation algorithm for the TSP-Library data set where $n = 439$ was then compared for $\alpha = 1$ & 2. Results showed that the algorithm was computationally faster for smaller values of p (i.e. 10, 20, 30 & 40) when $\alpha = 2$. Chen & Chen state that this is because the solution value is larger when $\alpha = 2$, which leads to fewer iterations and therefore less required computational time overall. However, the algorithm is computationally faster for larger values of p (i.e. 50, 60, 70, 80, 90 & 100) when $\alpha = 1$ (i.e. the classic $p-$centre problem). Chen & Chen suggest that as the solution to the $\alpha-$neighbour $p-$centre problem is always equal to or greater than the solution of the $(\alpha - 1)-$neighbour $p-$centre problem, the upper bound is higher which increases the number of potential facility locations and therefore the overall problem size. This interesting variation of the classic $p-$centre problem will be revisited in Chapter 7 where a relaxation-based algorithm is adapted to solve the $\alpha-$neighbour $p-$centre problem and tested against Chen & Chen's adapted classic relaxation algorithm.

The Conditional $p-$Centre Problem

As previously discussed, the $p-$centre problem can be classified as conditional. The

conditional problem wishes to locate p new facilities, given that q facilities already exist. The first conditional location problem can be found in Handler & Mirchandani (1979), and Minieka (1980) introduced conditional location problems on a network where both centres and medians were obtained. Drezner (1995) proposed the notation "(p, q) location problem" where p represents the number of new facilities to locate and q represent the number of existing facilities.

Chen & Handler (1993) modified a relaxation-based method (discussed previously in Section 2.2.4), which was initially developed to solve the unconditional $p-$centre problem (see Chen & Handler (1987)), for the conditional $p-$centre problem. Two adjustments were made to the original algorithm. The first one ensured that the demand points added to the subset were not within a certain distance of the fixed facilities, as this allowed the solution value to improve. The second adjustment modified how feasibility for the full problem was checked by calculating the largest distance from a demand point to its allocated facility where the fixed facilities are also included. When testing the algorithm, it was observed that it was very efficient for smaller problems but not for larger ones. Therefore, in an effort to improve this result, they introduced a maximum number of cuts when solving the problem. Once the maximum number of cuts was met, the algorithm started again with the best solution value found from the previous run and a new relaxed set of demand points. The authors found the new approach improved computational time such that optimal solutions were now obtained in less than 200 seconds. It is important to add that the idea of introducing a threshold to increase the chances of attaining the optimal solution inspired the scheme developed in Section 4.8, Chapter 4.

Berman & Simchi-Levi (1990) solved the conditional $p-$centre problem in a network using a distance matrix of demand points to all potential facility locations where the existing facilities are represented as one location. This meant a new facility could be located such that the solution value was improved. Berman & Drezner (2008) advanced this idea further by developing a modified distance matrix. Instead of finding the shortest distance from a demand point to any potential facility location, the shortest distance from a demand point to any existing facility was calculated instead. This is because in order to find an improved feasible solution, the distance from a demand

point to its closest existing facility must either decrease or stay the same (i.e. the demand point is allocated to an existing facility).

Drezner (1989) developed a binary algorithm for the conditional $p-$centre problem. Initially, the set of demand points, $\{i_1, i_2, \ldots, i_n\}$, are allocated to their closest existing facility. They are then sorted and listed in non-increasing order of Euclidean distance. Drezner states that there must be a value l such that the demand points $\{i_1, i_2, \ldots, i_l\}$ are not covered by the existing facilities within the optimal solution value, and therefore these are the demand points that need to be covered by the new facilities. A binary algorithm was used to solve a succession of $p-$centre problems for a subset of demand points in order to determine the value of l and obtain the optimal solution.

Chen & Chen (2010) developed Drezner's (1989) algorithm further by introducing their reverse relaxation algorithm, given in Chen & Chen (2009), to solve the subset of demand points more efficiently. The demand points were sorted as Drezner suggested, and for $k = 1, \ldots, l$, the subset $\{i_1, \ldots, i_k\}$ was optimally solved using the reverse relaxation algorithm which yielded a solution value Z. The algorithm terminated when Z was larger than the next furthest Euclidean distance in the ordered list, as this meant all demand points could now be covered by a facility (either existing or new) with this coverage value. The adaptations made to the reverse relaxation algorithm for the conditional $p-$centre problem consisted of adding one demand point to the subset at a time, and adding the demand point was the next to appear in the order list. Chen & Chen's algorithm for the conditional $p-$centre problem is explored further in Chapter 7, where it is compared to an enhanced version that incorporates the algorithm developed in Chapter 5.

2.3 Related Literature

This section will discuss the interesting literature that is related to this research, such as the discrete $p-$centre problem, metaheuristics for the $p-$centre problem and real life applications. Though the various methods and applications introduced in this section are not directly applied in this study, they still inspired the ideas and developments of

this research and so are worth mentioning.

2.3.1 The Discrete $p-$Centre Problem

Much like the continuous case, the discrete, or vertex, $p-$centre problem wishes to locate p facilities in a plane or network of demand points such that the maximum distance from a demand point to its nearest facility is minimised. However, unlike the continuous case, all the potential facility locations are pre-specified (e.g. on the demand points or on specific nodes). Although this research focuses on the continuous $p-$centre problem, this section will review the literature accomplished for the discrete case where ideas were adopted and adapted in this research for the continuous case.

As previously discussed, Hakimi (1964) was one of the first researchers to develop the idea of centres and medians in a network.

Dyer & Frieze (1985) gave a simple heuristic to solve the weighted discrete $p-$centre problem which consisted of a finite set of weighted points. The heuristic began by identifying the point with the greatest weight as the first facility site. The next facility is located at the demand point that is the greatest sum weighted distance from the previously selected facility location. This process is continued until p points are selected.

Plesník (1987) developed two heuristics where the first, described by the author as 'simple and natural', used a greedy heuristic to solve the discrete $p-$centre problem. First, a distance matrix was constructed for each demand point to each possible facility location. A sub-routine was used to find the minimum number of facilities required to cover the demand points within a given upper bound, and the heuristic repeated this sub-routine as the upper bound was increased to the next weighted distance in the matrix until the number of facilities required was $\leq p$. This heuristic was also modified such that it could heuristically solve the continuous case. As a distance matrix cannot be constructed for the continuous case, the modified heuristic determines all possible solution values (to be listed in non-decreasing order and set as upper bounds) by restricting the facility locations to $O(mn^2)$ points on the network (where n is the number of nodes and m is the number of edges). The heuristic then proceeds as normal to find

the minimum solution value needed such that all of the demand points are served by p facilities.

Ilhan & Pinar (2001) developed an algorithm for the discrete $p-$centre problem that solved many feasibility problems using a coverage distance found from the average of a lower and upper bound to the optimal solution value (much like the binary relaxation algorithm previously described). This algorithm was successful due to one key observation: if no feasible problem could be found for the corresponding linear programming problem (LP), then a feasible solution will not be found for the integer linear programming problem (ILP). In other words, the algorithm is split into two phases, where the first deals with solving the LP problem and the second with the ILP problem. Once feasibility has been found for the LP problem, the algorithm moves into the second phase. As the LP problem is easier to solve, the required computational time and effort was decreased, and therefore an efficient optimal algorithm for the discrete $p-$centre problem was created.

This important observation inspired more research into the discrete $p-$centre problem. During the second phase of Ilhan & Pinar's algorithm, the coverage distance is slowly increased by setting it as the next smallest distance between a demand point and a facility that is greater than the current coverage distance when no feasible solution is found for the sub-problem. The idea was adopted by Chen and Chen (2009) for their reverse relaxation algorithm that was used to solve the discrete and continuous $p-$centre problem (see Section 2.2.4).

Özsoy & Pinar (2006) also extended Ilhan & Pinar's algorithm to develop an exact algorithm for the capacitated vertex $p-$centre problem (i.e. each facility has a maximum capacity of demand points that it can serve). They formulated two additional sub-problems that could be embedded into the end of the algorithm to solve the capacitated $p-$centre problem. The first sub-problem considered the instance where the facilities' capacities were equal, and the second dealt with the case where the facilities' capacities varied.

Finally, Al-Khedhairi & Salhi (2005) developed three modifications for Ilhan & Pinar's

algorithm in order to improve its efficiency. Firstly, they noticed that when the algorithm moved from the first phase into the second phase (i.e. feasibility was found for the LP problem), updating the coverage distance as the lower bound would cause one wasted iteration as the LP formulation was found to be infeasible using this coverage distance. Therefore, instead they either updated the coverage distance as the upper bound, or did not change its value. Secondly, they observed that as the coverage distance is the average of the lower bound and the upper bound, it is not a 'real' distance between a demand node and a facility, and therefore not a potential solution value. Therefore, once the coverage distance was found, the value was adjusted to be the next minimum distance between a demand node and a facility greater than the coverage distance. Thirdly, Al-Khedhairi & Salhi proposed a new scheme for the second phase. Whereas Ilhan & Pinar found the next lower bound by selecting the next smallest distance between a demand node and a facility, Al-Khedhairi & Salhi suggest selecting the second smallest. Thus, their jump-based scheme involved missing out some values in order to reach the optimal solution quicker. This scheme is revisited later as it is used as a basis for an enhancement described in Section 5.3.3, Chapter 5.

In the same paper, Al-Khedhairi & Salhi modified a second algorithm, first proposed by Daskin (1995), to solve the discrete $p-$centre problem. Daskin's algorithm uses a search procedure, normally referred to as a binary search, to obtain the optimal solution. Much like Ilhan & Pinar's algorithm, the algorithm begins by selecting initial values for the lower and upper bound, and obtaining a coverage distance by calculating their average. The set covering problem is then solved for this coverage distance, and the minimum number of facilities needed, p'', is recorded. If $p'' \leq p$, then the optimal solution value is less than the coverage distance. The upper bound is set as the current coverage distance, a new coverage distance is calculated and the process begins again. If $p'' > p$, then the optimal solution value is greater than the current coverage distance. Therefore, the lower bound is reset as the current coverage distance, a new coverage distance is calculated and the process begins again. Thus at each iteration, a bisection between the upper and lower bound is made that pushes the bounds together until the optimal solution has been found (i.e. when the lower and upper bound are equal). This process of solving a succession of covering problems to ultimately solve the $p-$centre problem is adopted in Chapter 5.

Al-Khedhairi & Salhi (2005) proposed three modifications for Daskin's algorithm in order to reduce the number of iterations needed to find optimality. The first modification tightened the lower bound by starting with the p^{th} minimum value in the distance matrix. The second modification tightened the upper bound by finding the maximum distance in each row of the distance matrix, and taking the minimum of these. Finally, their last modification designed a more powerful binary search by introducing the numerical technique of the Golden Section. The modifications described for both algorithms reduced the number of iterations needed to find optimality, and therefore demonstrates that is it worthwhile adding simple enhancements to existing algorithms.

An efficient three-level heuristic used to solve the discrete $p-$centre problem was developed by Salhi & Al-Khedhairi (2010) which grouped facilities together within a predefined distance to form *neighbourhoods*. First, an initial solution is found using Cooper's multi-start heuristic. Then, the first level takes any facility and identifies the demand points encompassed by its covering circle and the demand points encompassed by covering circles of its neighbouring facilities (i.e. its neighbourhood). The location of this facility is then relocated to all these demand point sites. If an improved configuration is found, then it is kept and the customers served by the group of facilities are allocated to their closest facility within the neighbourhood. This is repeated for all facilities. The second level involves simultaneously moving two facilities to a customer location at a time, focusing on the largest covering circle. The third level used a perturbation step to avoid becoming trapped inside a local minimum. This step accepted infeasible solutions by allowing the solution to go above or below the required p facilities by a predetermined number q. The obtained solution is then used as an upper bound (UB) in the bisection method where the lower bound is then generated as $\beta(UB)$ where $\beta \approx 0.85$. The integration of heuristics into exact methods proved to be very promising.

This three level heuristic is particularly important to the work discussed in Chapter 6. It inspired the development of a matheuristic that incorporates the idea of grouping facilities together to form neighbourhoods, and, as previously stated, adopts the logic of focusing on decreasing the size of the largest covering circle in order to improve the solution value.

Other research for the discrete $p-$centre problem includes Dantrakul & Likasiri (2012), who developed an algorithm for the capacitated $p-$centre problem and tested it against the one devised by Albareda-Sambola *et al* (2010). Chen & Chen (2010) adapted the reverse relaxation algorithm to solve the discrete and continuous conditional $p-$centre problem, see Section 2.2.5. Finally, Averbakh & Berman (1997) studied the probabilistic $p-$centre problem, that is, finding the optimal solution for a problem that incorporates future values (e.g. change in population) into the formulation. They approached the problem by first modeling it using probability distribution to reveal possible scenarios or changes in population over the network, and then using the weight range over the population by calculating the 'best of the worst case scenario' using the largest weights for each demand points.

2.3.2 Metaheuristic Methods for the $p-$Centre Problem

Many powerful heursitic, or metaheuristic, methods have been developed in order to find a tight feasible solution for large $p-$centre problems where obtaining optimal solutions was not possible. This section will discuss several different metaheuristics developed to solve either the discrete or the continuous $p-$centre problem.

As previously stated in Section 1.5.2, Chapter 1, Davidović *et al* (2011) proposed a metaheuristic method called Bee Colony Optimization (BCO) and an improved method ($BCOi$) to solve the discrete $p-$centre problem, and Kennedy & Eberhart (1995) developed a new metaheuristic called Particle Swarm Optimization (PSO) which was developed by Khodary & Tharwat (2013) to solve the continuous $p-$centre problem.

Irawan *et al* (2016) developed two meta-heuristics used to solve large discrete $p-$centre problems and conditional $p-$centre problems. Both meta-heuristics used the aggregation method to find an approximate solution to the problem. In other words, a problem of size n was transformed into a set of sub-problems of size n', where $n' << n$. This method was used as a tool, alongside exact methods and the Variable Neighbourhood Search algorithm, to find a good approximate solution initially. Both metaheuristics, as well as the exact methods, were then adapted such that they can be used to solve the conditional $p-$centre problem. Similar adjustments can be seen in Chapter 7, as

the algorithm developed in Chapter 5 was modified to solve the conditional continuous $p-$centre problem.

In more recent work, Elshaikh *et al* (2016) developed another perturbation-based heuristic for the continuous $p-$centre problem. The perturbation method in this instance allowed the number of facilities p to go over or under by a set amount q, thus making the solution infeasible at certain points in the algorithm. This perturbation method was first suggested by Salhi (1997) for the $p-$median problem, and adapted by Salhi & Al-Khedhairi (2010) for the discrete $p-$centre problem. The heuristic has three main stages that are a) adding q facilities, b) dropping q facilities and c) swapping facilities. The authors developed and tested two variants of this perturbation algorithm, namely GRADPERT and STRONGPERT. The GRADPERT algorithm adds one of the extra q facilities at a time before using a local search, whereas the STRONGPERT adds all q facilities altogether before using the local search. STRONGPERT was also enhanced using learning and was found to perform the best.

2.3.3 Real Life Applications

This section provides a brief review of some real-life case studies for the $p-$centre problem and its related location problems. It therefore demonstrates the practicality of the $p-$centre problem solution, and the impact that a tight solution can make in a real-life scenario where emergency facilities must be located efficiently to save lives.

Richard *et al* (1990) proposed an interesting paper that compared the set covering model, the $p-$median model and the $p-$centre model. The purpose of the paper was to use the three models to locate fifteen fire stations in the Belgian rural province of Luxembourg, and compare these results to the locations in real life to see if either model could yield an improvement. They highlighted that this area was very rural, which made the study rare as most research selected urban areas. The data collected over a five year period from the existing fifteen fire stations found a total of 908 demand points that included villages, sparsely populated hamlets and uninhabited areas where accidents occurred regularly such as roads. The 99 potential facility locations were identified in a similar way. The results showed that the $p-$median gave the best results in terms of minimising the maximum travel time and mean time. The results

were compared with the coordinates for the real fifteen fire stations and showed that the real locations could be improved by using both the $p-$median or the $p-$centre solution.

Pacheco & Casado (2004) presented a real health case study where a small number ($p < 10$) of health resources, such as geriatric and diabetic health care clinics, were located in the area of Burgos, Spain. As this is a rural area with a dispersed population, it is important to locate emergency facilities in an efficient way so that medical emergencies can be dealt with quickly. Both the discrete $p-$centre problem and the maximal covering problem were solved for this real life case study using a scatter search, and the results were compared to that of the Variable Neighbourhood Search (VNS). On average, the scatter search yielded the same quality of result as the VNS, but it took less computational time.

As previously discussed in Section 1.5.2, Chapter 1, Kaveh & Nasr (2011) modified a harmony search to solve the following real life problem. In the city of Isfahan, ten new bicycle stations needed to be established each year given that eight bicycle stations were successfully located in the city during the previous year. Therefore, the problem is the conditional $p-$centre problem, where ten stations need to be located given that eight already exist. Firstly, the unconditional problem was solved where $p = 8$, followed by the conditional $10-$centre problem where the location of the first eight stations were accounted for. A similar approach is used in this study when the relaxation-based algorithm developed in Chapter 5 is incorporated into a known algorithm used to solve the conditional $p-$centre problem, see Chapter 7.

Wei *et al* (2006) developed Suzuki & Okabe's (1995) heuristic for the constrained $p-$centre problem (see Section 2.3.4) and applied it to the real life case study of locating 25 service facilities in Dublin, Ohio. They compared their algorithm to Suzuki & Okabe's in terms of solution value and the computational time required for each iteration and overall. Suzuki & Okabe's algorithm required less computational time (1.2 hours), however two facilities were located outside of the city border. Wei *et al*'s algorithm required 2.9 hours to solve the problem, but all facilities were located inside the city border and so gave a feasible solution. More information on their enhanced algorithm can be found in Section 2.3.4.

Murray, O'Kelly & Church (2006) located warning sirens in Dublin (Ohio) where both the number of facilities was minimized and the entire solution space was covered. They represented the demand points in four ways (i.e. point representation, modified point representation, regional representation and area representation) and compared the results of each representation type on its efficiency and effectiveness. They found that when the demand point were represented in the traditional way (i.e. point representation), $14 - 27$ sirens were required. However, this solution did not cover the entire solution space. In other words, anyone stood in the uncovered areas at the time of the sirens sounding would not be able to hear them. When using the modified point representation method, it was found to require $19 - 26$ sirens where $8.35\% - 1.16\%$ of the plane remained unserved. The strongest representation types were regional and area representation, as 26 sirens where required to cover the entire solution space. However, a weakness of area representation is that there may exist wasted coverage as certain areas were covered that did not need to be (i.e. outside of the solution space boundary), or some areas were covered by more than one siren.

Murray & Wei (2013) applied the set covering problem to a real life scenario where the demand points where represented in two different ways, similar to the representations described above. The first approach was the traditional point representation, and the second approach defined the customer as a polygon area rather than a single point. In other words, if a facility serves a customer, then this means it must serve the entire polygon area. This approach may be useful when locating facilities that need to cover whole areas rather than specific points, such as the instance of emergency sirens mentioned above or locating street lamps to ensure that a whole street is lit. Two real life applications were investigated using this point representation method. The first minimised the number of emergency warning sirens that need to be placed in Dublin, Ohio, and the second application sited the minimum number of fire stations in Elk Grove, California. These two applications showed this method to be robust and efficient as both had a clear result on the number and location of the facilities.

Lu (2013) presented a case study based on the earthquake in Taiwan where the author solved the $p-$centre problem when travel time and demand for each facility differs.

The earthquake, that was measured at 7.3 on the Richter scale, caused 2500 deaths, 8000 injuries and damaged 39,000 buildings. Therefore, in instances such as these, it is important to accurately locate the limited number of urgent relief distribution centres (URDC) to help with the casualties. As the URDCs are located in established buildings, such as schools and warehouses, the problem was formulated as a discrete $p-$centre problem. Lu developed a simulated annealing (SA) heuristic that dealt with uncertain travel times and demand at each facility. The SA-based heuristic was compared to exact solutions found using complete enumeration for benchmarking purposes, and was found to give good results. It then proceeded to locate two URDC's to aid casualties after the earthquake from a selection of seven URDCs amongst fifty-one demand points. For more information on the SA heuristic, see Salhi (1998), (2006).

2.3.4 Other Inspirational Literature

This section will discuss other interesting literature that was not directly applied to this research. This includes briefly reviewing the constrained $p-$centre problem, the maximal covering problem and solving the continuous $p-$centre problem in networks and in rectangular areas.

The Constrained $p-$Centre Problem

The $p-$centre problem is classified as constrained if there are certain forbidden areas in the solution space where the facilities cannot be situated. Research in this area has been conducted for the discrete case (such as Murray, O'Kelly & Church (2006)), and this method is also mentioned for the continuous case, see the chapter based on full covering location problems by Plastria (2002).

Suzuki & Okabe (1995) developed a Voronoi diagram heuristic for the continuous $p-$centre problem. The algorithm initially located p facilities at random, and constructed p Voronoi diagrams based on these p facilities. The current facility locations were then replaced by the centre of each Voronoi diagram, and this was repeated until the configuration of the facilities' locations didn't change. As the final solution value was highly influenced by the initial starting locations, the authors suggested repeating the process many times and to obtain a tight solution. Suzuki & Drezner (1996) developed this algorithm further by creating a 'finishing-up' algorithm that decreased the

size of the maximum covering circle and simultaneously increased the size of a smaller covering circle to maintain coverage. The authors found that this converged the solution faster. It is important to note that this Voronoi-based heuristic shares similarities with the matheuristic proposed in Chapter 6, as Voronoi-based algorithms inspired its development.

As previously mentioned in Section 2.3.3, Wei *et al* (2006) highlighted several areas of possible difficulty when applying Suzuki & Okabe's Voronoi heuristic to a real-life case study due to the following three assumptions that a) the overall region is rectangular or square, b) all Voronoi polygons are simple and c) facilities can be located anywhere. The authors therefore developed the algorithm such that it could be used to solve the constrained $p-$centre problem for real-life data by finding the constrained minimal covering circle ($CMCC$) when calculating the centre of the Voronoi polygon. Firstly, the subset was treated as unconstrained and so the minimum covering circle (MCC) was found. If the solution to the MMC was not in a forbidden region, this was taken as the solution. However, if the centre of the MMC lay in a forbidden region, two repair mechanisms were used to find the best $CMCC$. The first consisted of constructing the convex hull for the region and finding covering circles formed from a point on the convex hull and its closest point on each line segment. The second method investigated all pairs of points in the convex hull. The solution circle was taken as the smallest of all possible $CMCC$'s.

Davoodi *et al* (2011) also proposed a Voronoi-based algorithm that was similar to the approach suggested by Wei *et al* (2006). They began with a simple, but effective, algorithm that consisted of allocating demand points to their closest facility, constructing the Voronoi diagrams based on these facilities, computing the MCCs and checking if the centre of the diagrams lay outside the forbidden regions. If they did not, the process was simply repeated. They found this algorithm to be fast as it did not have many iterations. However, a possible weakness was highlighted as they stated that "the high convergence in the heuristic algorithm usually causes them to become trapped in the local optima." (Davoodi *et al* (2011): 3323). They suggested an improved heuristic which moved the centres of the smaller circles closer to the centre of the largest circle(s) in an endeavour to avoid becoming trapped in a local optimum. This created

a computationally more time-consuming algorithm but with improved results.

The Maximal Covering Problem

The maximal covering problem, first defined by Church & Revelle (1974), wishes to locate p facilities such that the number of demand points served by each facility is maximised. Alternatively, the aim is to minimise the number of uncovered demand points. A relation can be seen between the maximal covering problem and the multi-level set covering problem discussed in Section 2.2.5, as both location problems deal with maximising coverage.

Mehrez & Stulman (1982) investigated the discrete maximal covering problem (i.e. a facility can only be located on a pre-specified point) for both the single facility problem and the multiple facility problem. In both instances, a facility was located at the centre of a covering circle with radius R. For the single facility problem, if a circle with radius R was placed at every demand point with no intersections between the circles, then the final facility could be placed at any demand point. However, if two or more circles intersected, then the optimal solution was found in the intersection where most demand points were covered. The multiple facility problem could be solved with either complete enumeration or dynamic programming.

Church (1984) developed the maximal covering location problem by introducing a new problem called the planar maximal covering problem ($PMCE$). This is where the facilities can be located anywhere on the plane (i.e. a continuous problem). First, the author obtained the finite set of potential facility locations, and then found the set of circle intersection points (CIPs). The author then proved that at least one optimal solution of the $PMCE$ has all facilities' locations lying in the CIPs. He enhanced the algorithm further by excluding circles from the potential set of facilities that were dominated by another covering circle, which lead to a reduced set of circle intersection points and thus decreased the problem size.

The Continuous $p-$Centre Problem in Networks

Hakimi (1964) was the first to formulate the $1-$centre problem in a network, and proposed a method of finding the absolute centre of a graph (i.e. a facility can be located

anywhere on the edges or vertices of the graph). If there are l edges to a graph and n vertices, finding the absolute centre involves solving l 'min-max' problems. However, as this requires $n!(n-1!)$ calculations, it would require a large amount of computational time for very large problems.

Handler (1978) presented a simple algorithm to locate the absolute centre in a tree network, and extended this algorithm to solve the absolute $2-$centre problem. The idea was to find the absolute $1-$centre of the graph first using established methods. A cut is made in the tree on the edge that the absolute $1-$centre point is located on, splitting the tree network into two smaller sub-graphs. The absolute $1-$centre was found for both sub-graphs to find a solution for the $2-$centre problem.

Minieka's (1970) proposed an interesting algorithm for the continuous $p-$centre problem in a graph where $p > 1$. He uses minimal set covering to solve the $p-$centre problem optimally. In other words, he found the minimum number of facilities required so that all the demand points were covered. His algorithm begins with an arbitrary first choice of p facilities (located anywhere on an edge or vertex) as an initial solution, yielding a solution value Z. The aim is to find another set of p locations for the facilities such that the solution value $< Z$. The SCP is solved for the value Z, and this yields the minimal number of facilities needed to serve all the points. If the minimum number of facilities needed is $< p$, then the solution can be improved. Therefore, Z is set as the new solution value and the process begins again. However, if the minimum number of circles is $\geq p$, the solution cannot be improved and so Z is the optimal solution value. Thus, Minieka's algorithm decreases the value of Z until optimality is reached. This method is important as it has been advanced and adapted in more recent research to solve large discrete and continuous $p-$centre problems, some of which have previously been discussed in Section 2.3.1.

Daskin (1995) developed a simple but effective algorithm to find the absolute $1-$centre for a weighted tree network. The method starts by finding the centre for two weighted nodes by using their weight ratios. He then uses the same idea to solve the problem with three nodes, where the centre of the first two weighted nodes are treated as one node. This method of solving two at a time is then used to solve a larger network using

a weighted distance matrix.

Solving the Continuous $p-$Centre Problem using Rectangular Areas

Drezner & Hamacher (2001) describe the process for finding the smallest rectangular space that covers all the points (rather than the smallest covering circle). The technique is to find the smallest diamond that covers a set of given demand points. The four corners of the diamond are labelled LT, RT, LB and RB for the left-top corner, right-top corner, left-bottom corner and right-bottom corner respectively. The process begins with an initial rectangle that covers a subset of demand points. The rectangle is then increased in size by pushing down on the LT and RT corners, and up on the LB and RB corners until it reaches an uncovered demand point in both directions. The process continues until all the demand points are covered and the optimal location for the facility lies at the centre of the diamond.

Wesolowsky (1972) also studied solving the $p-$centre problem using rectangular areas. He gave several examples where he compared solving small problems using linear programming or graphical methods. The method of using rectangular areas was also adopted by Drezner & Wesolowsky (1993), who used it to solve maximin problems.

More information on solving the $p-$centre problem using rectangular distances can be found in Drezner (1987).

Vijay's Exact Method

Similar to Drezner's (1984) exact method, Vijay (1985) also proposed an algorithm that solved the $p-$centre problem using geometric properties. He described a way to reduce the problem size by finding dominated circles. If a subset of demand points was covered by a non-dominated circle, this subset was called a *maximal subset* and the circle was kept as a potential facility location. He used a rotation technique to find all the maximal subsets. Each pair of points created a circle of radius Z, where an initial Z was found using a locate-allocate heuristic. The circle was then rotated 360° around one of the points on its circumference, and each time a demand point entered or left the circle it was recorded. Thus, it could be seen if certain circles held the same, or the same and more, demand points to the others. Once all maximal subsets of a

given value Z was found, the problem could then be solved using an ILP method. Similarly to Drezner's maximal circles method, the problem can now be solved in less computational time and effort as the number of the potential facilities has greatly decreased. Vijay states that the setup for this algorithm is where computational time is at its highest, however, once the setup is complete the problem is solved very quickly. Therefore, a balance between the computational time required to set up the algorithm and to solve the problem must be found.

2.4 Summary

This chapter has given a detailed literature review for both the studies directly applied to this research, and for the related ones. The first section highlights the studies that were utilized in this thesis (such as the exact methods that were investigated and enhanced), the heuristic methods that were incorporated into optimal algorithms and the related locations problems. The second section dealt with the important and relevant literature that, although not directly used in this research, gave key elements that contributed to our research. This included methods used to solve discrete $p-$centre problem as well as and powerful metaheuristics developed for complex, large location problems.

The next chapter will discuss and formulate the main location models used in this research, and will also examine the initial investigative work that was conducted.

Chapter 3

Methodology and First Investigative Results

3.1 Introduction

This chapter begins by studying the formulations of the relevant location problems to this research such as the set covering problem, the vertex $p-$centre problem and the continuous $p-$centre problem. As this thesis focuses on the latter, this issue will be discussed in greater depth. The chapter will also investigate heuristic methods, and observe how their solution quality can be used as a stepping stone to obtain the optimal solution. Two heuristics, namely the classical Cooper's multi-start in the discrete space, and its extended version, namely Cooper's alternate locate and allocate algorithm, are compared. Furthermore, a new scheme that attains all circles formed from three demand points is described and its computational efficiency evaluated.

3.2 Location Models

As previously stated, location models can be classified under two main categories, namely discrete models and continuous models. We recall that the potential facility sites for the discrete models are fixed positions, or nodes on a network, whereas in the continuous case there are no restrictions on site location which means a facility can be situated anywhere on the network or in the plane.

We shall now introduce the formulations of the location problems that were used in this research. As this thesis will be focusing on the absolute, or continuous, $p-$centre problem, for completeness we shall also describe two discrete $p-$centre problem-based formulations that are incorporated into this research. Three formulations used to solve the continuous $p-$centre problems are then presented, with the final one being the one that will be extended upon. We shall begin by describing the Set Covering Problem,

which is used as one of the sub-problems that is solved optimally as part of solving the $p-$centre problem. For more information on discrete models, see Daskin (1995) and references therein.

3.2.1 The Set Covering Problem

The Set Covering Problem (SCP) aims to minimise the number of facilities, whilst ensuring that all the customers (demand points) are served within a threshold distance (e.g. $R = 4$ miles) or travel time (e.g. 10 minutes). Here we refer to the location of a facility j as the centre of circle C_j. The SCP formulation is given as follows.

Firstly, let us define the binary matrix, $A'_{i,j}$.

$$A'_{i,j} \begin{cases} 1 \text{ if } d_{i,j} \leq R \text{ (i.e. point } i \text{ can be covered by the } j^{th} \text{ facility with radius } r_j \leq R), \\ 0 \text{ else.} \end{cases}$$

with

$d_{i,j}$: Euclidean distance from demand point i to the centre of circle C_j, $i \in I, j \in J$;

R: threshold imposed.

$$\text{Minimise} \qquad \sum_{j \in J} x_j \tag{3.1}$$

$$\text{subject to} \qquad \sum_{j \in J} A'_{i,j} x_j \geq 1 \quad \forall i \in I, \tag{3.2}$$

$$x_j \in \{0, 1\} \quad \forall j \in J, \tag{3.3}$$

where:

I: the set of demand points, where $i = 1, \ldots, n$;

J: the set of facilities (potential circles with their corresponding centres), where $j = 1, \ldots, m$;

$$x_j = \begin{cases} 1 \text{ if a circle } C_j \text{ is selected,} \\ 0 \ \textit{else.} \end{cases}$$

The objective function (3.1) minimises the number of facilities (covering circles) opened. Constraint (3.2) ensures that each customer is covered by at least one facility and con-

straint (3.3) represents the binary decision variable x_j.

This model will be embedded into the relaxation-based algorithm that will be presented in Chapter 5.

3.2.2 The $P-$Centre Problem

The $p-$centre problem seeks to find the best location of p facilities in a network or plane to serve n demand points such that the maximum distance or time from a demand point to its closest facility is minimised. In other words, the $p-$centre problem wishes to reduce the outcome of the worst case scenario. For example, one may wish to find the location of p hospitals in a town to serve n residential homes, such that the maximum distance from any residential home to its closest hospital is minimised. This strategy aims to reduce the risk of accidents of emergencies becoming more complicated or serious.

The formulations of the two different model types (i.e. discrete and continuous) will now be described.

(a) The Vertex $P-$Centre Problem

The first model for the vertex $p-$centre problem uses an integer linear programming (ILP) formulation, whereas the second model is based on the SCP formulation.

(i) The ILP Formulation

The ILP formulation for the vertex $p-$centre problem is shown below.

$$\text{Minimise} \qquad W \tag{3.4}$$

$$\text{subject to} \qquad \sum_{j \in J} Y_{i,j} = 1 \quad \forall i \in I, \tag{3.5}$$

$$\sum_{j \in J} x_j = p, \tag{3.6}$$

$$Y_{i,j} \leq x_j \quad \forall i \ \in I, \forall j \in \ J, \tag{3.7}$$

$$W \geq \sum_{j \in J} d_{i,j} Y_{i,j} \quad \forall i \in I, \tag{3.8}$$

$$x_j, \ Y_{i,j} \in \{0,1\} \ \forall i \in I, \ \forall j \in J. \tag{3.9}$$

where

W : is the maximum distance between a facility and a demand point;

$$Y_{i,j} \begin{cases} 1 \text{ if demand point } i \text{ is assigned to facility } j, \\ 0 \ \textit{else.} \end{cases}$$

All other definitions are as previously given.

The objective function (3.4) minimises the maximum distance between any demand point and its nearest facility. Constraint (3.5) guarantees that all demand points are assigned to a facility, and constraint (3.6) ensures that exactly p facilities are located. Constraint (3.7) imposes that demand points are only assigned to another demand point if a facility is located on it. Constraint (3.8) ensures that W is the maximum distance between any demand points and its allocated facility. Constraint (3.9) refers to the binary decision variables x_j and $Y_{i,j}$.

Note that if the demand points are weighted (the case above assumes they are of equal weight) constraint (3.8) would be adjusted to

$$W \geq w_i \sum_{j \in J} d_{i,j} Y_{i,j} \quad \forall \ i \in I \tag{3.10}$$

where w_i indicates the weight of node i.

(ii) The SCP-Based Formulation

The discrete $p-$ centre problem can be efficiently solved using the SCP-based method, which has already been briefly described in Section 2.3.1, Chapter 2. This method iteratively updates the range (LB, UB) where LB and UB refer to the lower and upper bound of the solution value W respectively. A coverage distance $CD = \frac{LB+UB}{2}$ is found, and the SCP is then solved with the coverage distance treated as an upper bound to the solution value. If the number of facilities needed to solve the SCP is $> p$, then CD is infeasible for the $p-$center problem and so must be a lower bound. The lower bound LB is updated so that $LB = CD$. Else, the number of facilities needed to solve the SCP is $\leq p$, thus giving a feasible solution to the problem. The upper

bound UB is updated such that $UB = CD$. The process is repeated until (LB, UB) does not contain any further coverage distances and so $W = UB$. For more details, see Salhi and Al-Khedhairi (2010).

This method is important as it is incorporated into our research in Chapter 5 when enhancing a well known relaxation-based algorithm.

(b) The Continuous $P-$Centre Problem

The continuous $p-$centre problem can be formulated mathematically in several different ways. Below, we see the formulation of the Euclidean unweighted $p-$centre problem as stated by Chen & Chen (2009).

$$Minimise\ _{X_1 \ldots X_p}\ \{Max\ _{i \in I}\ [Min\ _{1 \leq j \leq p}\ d_{i,j}]\} \tag{3.11}$$

where:
$X_j = (x_{j_c}, y_{j_c})$ is the location of facility j with its Cartesian coordinates for $j = 1, \ldots, p$.
All other definitions are as previously given.

Secondly, the problem can also be formulated as a non-linear mathematical programming formulation as follows.

$$\text{Minimise} \qquad W \tag{3.12}$$

$$\text{subject to} \qquad W \geq d_{i,j} Y_{i,j} \quad \forall i \in I,\ j = 1, \ldots, p, \tag{3.13}$$

$$\sum_{j=1}^{p} Y_{i,j} \geq 1 \quad \forall i \in I, \tag{3.14}$$

$$Y_{i,j} \in \{0, 1\}\ \forall i \in I, j = 1, \ldots, p, \tag{3.15}$$

$$X_j \in \mathbb{R}^2\ j = 1, \ldots, p. \tag{3.16}$$

The objective function (3.12) minimises the maximum distance between any demand point and its nearest facility. Constraint (3.13) ensures that W is the maximum distance in the solution and constraint (3.14) guarantees that every demand point is covered by at least one facility. Constraints (3.15) and (3.16) refer to the binary and real decision variables respectively. Note that this formulation is non-linear as $d_{i,j}$ rep-

resents the Euclidean distance from demand point i to facility j.

The third, and main formulation used in this research, is based on the SCP. For clarity, we shall refer to this particular formulation from now on as the classic $p-$centre problem formulation, abbreviated to For_{pc}.

Firstly, let us define the binary matrix, $A_{i,j}$.

$$A_{i,j} \begin{cases} 1 \text{ if } d_{i,j} \leq r_j \text{ (i.e. point } i \text{ can be covered by the } j^{th} \text{ facility with radius } r_j), \\ 0 \text{ else.} \end{cases}$$

$$For_{pc}: \text{Minimise} \quad W \tag{3.17}$$

$$\text{subject to} \quad \sum_{j \in J} A_{i,j} x_j \geq 1 \quad \forall i \in I, \tag{3.18}$$

$$\sum_{j \in J} x_j = p, \tag{3.19}$$

$$W \geq x_j r_j \quad \forall j \in J, \tag{3.20}$$

$$x_j \in \{0, 1\} \; \forall j \in J, \tag{3.21}$$

where all definitions are as previously given.

The objective function (3.17) wishes to minimise the maximum distance between any demand point and its closest facility. Constraint (3.18) ensures that every customer (i.e. demand point) is served by at least one facility. Constraint (3.19) states that only p facilities are located and constraint (3.20) guarantees that W is the maximum distance between any demand point and its allocated facility. Constraint (3.21) refers to the binary decision variable.

The next section will now discuss the initial research conducted that is incorporated into the future research.

3.3 Initial Research

In our preliminary experiments, several ideas were investigated with a view to minimise either the computational time or computer memory required for large continuous $p-$centre problems so that they may be solved optimally. Two experiments that helped guide and shape the main body of research conducted in this thesis will now be described. In brief, the first idea explored the integration of heuristic solutions within optimal methods to aid in obtaining the optimal solution, and the second explored a simple but exciting new idea to save computational time when creating the potential facility locations formed from three demand points.

3.3.1 Preliminary Experiments with Heuristic Methods

A heuristic method gives an approximate, or good, solution, but not necessarily an optimal one. However, as stated previously, a heuristic solution can be used as an initial upper bound that can be incorporated into optimal techniques to tighten the search and hence obtain the optimal solution relatively quickly.

For the case of the continuous $p-$centre problem, finding an upper bound to the optimal solution means that any covering circle (from the set of potential facility locations) with a radius larger than the upper bound can be discarded, as a solution has been obtained with p circles whose radii are equal to or smaller than the upper bound. Once these larger circles have been removed from the search, the number of calculations and overall computational time will decrease. This increases the chances of optimally solving larger location problems in a reasonable amount of time. Therefore, the better the quality of the heuristic solution, the tighter the upper bound found and therefore there is an increased chance of optimally solving larger problems.

Investigative work was completed for two different heuristics to help solve large continuous $p-$centre problems, namely the Cooper's multi-start discrete method ($MSDA$), also known as Cooper's discrete locate-allocate, and Cooper's multi-start method in the continuous space ($MSEHA$) as previously discussed in Section 2.2.2, Chapter 2. As the $MSEHA$ requires an established method to optimally solve the $1-$centre problem, the Elzinga & Hearn (1972) algorithm is used here, and is referred to as EHA.

The full description of EHA is given in Algorithm 1. For clarity, the full algorithm for the $MSEHA$ is found in Algorithm 2 as it is used widely in this research. The $MSDA$ shares many similarities to the $MSEHA$ algorithm, and so, for completeness, the steps of this algorithm that differ from $MSEHA$ can be found in Algorithm 3.

Algorithm 1. *The EHA Algorithm*

1. *Select two points P_1 and P_2 .*
2. *Construct a circle whose diameter is the line joining P_1 and P_2. If this circle encompasses all the points, stop. If not, go to Step* 3.
3. *Select a point P_3 outside the circle made from P_1 and P_2.*
4. *Determine if points $\{P_1, P_2, P_3\}$ form an acute, right-angled or obtuse triangle.*
   (i) Right-angled or obtuse triangle: Take the two points furthest away from each other and go back to step 2.
   (ii) Acute triangle: Construct the circle passing through these three points. If this circle encompasses all the demand points stop. Else, go to Step 5.
5. *Select a point P_4 outside of the circle, and let H be the point from $\{P_1, P_2, P_3\}$ that is furthest away from P_4. Extend the diameter through the point H so the plane is divided into two halves. Select a point L from $\{P_1, P_2, P_3\}$ that is in the half plane opposite to the point P_4. Set $P_1 = H$, $P_2 = L$ and $P_3 = P_4$ and go to Step 4.*

Algorithm 2. *The MSEHA Algorithm*

1. *Select the number of multi-starts, $MulStart$. Set $Mul = 0$.*
2. *While $Mul < MulStart$ do:*
   (a) *Set Mul=Mul+1; Select any p points from the m customer sites or on the plane to be temporary facility locations.*
   (b) *Allocate all the demand points to their closest temporary facility. This creates p clusters.*
   (c) *For each cluster, find the optimal solution for the single facility problem using Algorithm 1.*

(d) Reset the location of the facilities as the p circle centres found for each single facility problem.

(i) If the configuration of the facilities change, go back to Step 2 (b).

(ii) Else, take the radius of the largest circle as the solution value. Go back to Step 2 (a).

3. Take the smallest solution value found as the final value. Stop.

Algorithm 3. *The MSDA Algorithm*

1. Steps 1-2 (b) in Algorithm 2.

2. (c) In each cluster, find the furthest Euclidean distance between any two points. This yields p distances.

(d) Take the largest of the p distances as the solution value. Go to Step 2(a).

3. Take the smallest solution value as the final value. Stop.

Note that as Step 2 (c) in Algorithm 3 is time consuming and Step 2 is repeated several times, the $MSEHA$ requires more computational time than the $MSDA$.

Comparing the Heuristics

The two heuristics were tested on a generated data set where $n = 100$ (see Figure 3.1). The algorithm was written on a HP Elitebook 8570w with 12 GB of RAM. Table 3.1 shows the solution value found using each heuristic, and this is compared to the optimal solution to see how far each heuristic deviated from the exact answer.

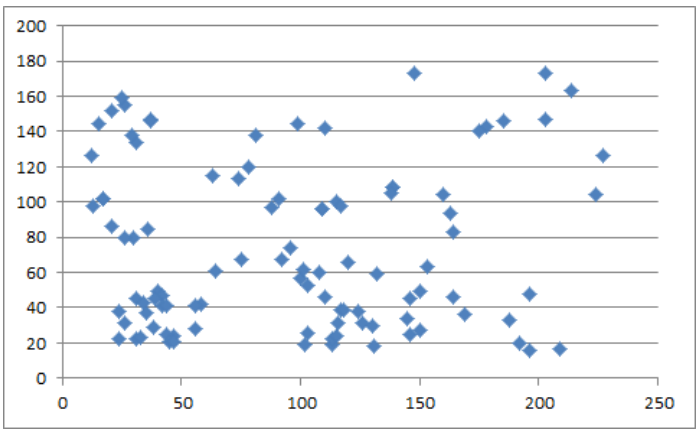


Figure 3.1: Distribution for the generated data where $n = 100$

Note that the deviation (%) was calculated by:

$$\frac{H_p - O_p}{O_p} \times 100, \tag{3.22}$$

where H_p and O_p refer to the heuristic and optimal solution for a given p respectively.

Table 3.1 is organised in the following way. The first column, labelled 'p', shows the number of facilities located. The second column, titled Z^*, displays the optimal solution for the corresponding p value. The next two columns displays each heuristic's solution value, Z_H, which could be used as an initial upper bound for optimal methods. To test the quality of the solution given for each heuristic, the last two columns show the percentage deviation that the heuristic solution is from the optimal solution. Figure 3.2 is also given to illustrate the solution values found for each heuristic compared to the optimal solution.

		Z_H		Deviation (%)	
p	Z^*	MSDA	MSEHA	MSDA	MSEHA
2	85.566	113.071	85.566	32.1	0.00
5	49.649	79.158	49.649	59.4	0.00
10	29.921	55.000	29.921	83.8	0.00
15	21.319	44.553	24.910	109.0	16.8
20	16.101	42.426	21.319	163.5	32.4
25	13.210	30.870	16.867	133.7	27.7
30	11.161	28.231	14.603	152.9	30.8
Average	**32.418**	**56.187**	**34.689**	**104.9**	**15.4**

Table 3.1: Results for the Heuristic Investigation where $n = 100$

It is clear to see that the solution value found using the $MSDA$ deviates much further from the optimal solution compared to the solution value found using the $MSEHA$. This is the expected result, as the $MSDA$ can only choose possible facility locations from a discrete set of points. This restricts where the facilities can be located, thus restricting the overall solution value. The $MSEHA$ does not have these restrictions, however, as facilities can be located anywhere in the plane. This therefore allows the solution value to be much tighter. It is worth noting that the solution of each run of the $MSDA$ could be used as an initial solution for the $MSEHA$, rather than starting with p random points, in order to obtain an even tighter upper bound.

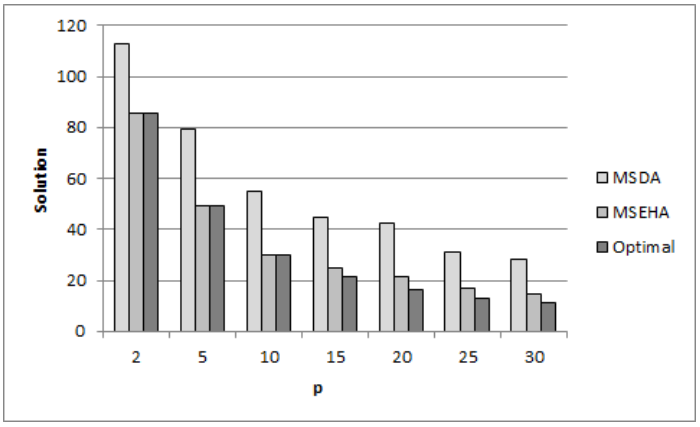


Figure 3.2: Heuristic solution values compared to the optimal solution value

Although the importance of finding a good heuristic solution is practically demonstrated in Chapters 4 and 5, it is especially relevant in Chapter 6. This is because this chapter creates a matheuristic by combining the $MSEHA$ mentioned above with the exact algorithm developed in Chapter 5.

Note that the generated data set can be collected from the author or accessed from the Centre for Logistics and Heuristic Optimisation (CLHO (2016)) website http://www.kent.ac.uk/kbs/research/research-centres/clho/datasets.html by following the links to 'continuous data sets' and then to 'random $p-$centre'.

3.3.2 The Upper Bound's Effect on the Duality Gap

During this research, the program IBM ILOG CPLEX (referred to as CPLEX) was used to solve the problems optimally. CPLEX is a well known commercial ILP solver that, as previously discussed, uses a Branch & Cut method to optimally solve location problems.

CPLEX allows a parameter that controls the termination of the program based on the value of the duality gap to be manually inserted into the code. In other words, once the duality gap reaches a particular value of our choosing (say, 10%), CPLEX stops and reports the best solution value if found.

Initial investigative work was completed that explored how using the heuristic's solution value as an initial upper bound affected the duality gap value (DV). The computational time for the duality gap to reach 0% (i.e. the optimal solution) using

the respective upper bound was recorded to reveal the affect a tight upper bound has on the overall computational time. The upper bounds were set as the $MSDA$ and $MSEHA$ solution values given in Table 3.1, and the results found are displayed in Tables 3.2 and 3.3 which are both organised in the following way. The first two columns are organised the same as Table 3.1. The remaining three columns display the results for the $MSDA$ and for the $MSEHA$ respectively. That is, the heuristic solution, Z_H, from the respective algorithm that was used as an upper bound, the number of circles (i.e. potential facility locations) inputted into CPLEX and the computational time taken to find the solution value where $DV = 0\%$ respectively.

p	Z^*	Z_H	# Circles	CPU Time (secs)
2	85.566	113.071	43928	62.45
5	49.649	79.158	35624	58.23
10	29.921	55.000	18641	23.60
15	21.319	44.553	10819	13.12
20	16.101	42.426	9567	14.44
25	13.210	30.870	3851	4.36
30	11.161	28.231	3130	3.90
Average	**32.418**	**56.187**	**17937**	**25.73**

Table 3.2: Results for the Duality Gap Investigation for $MSDA$ ($DV = 0\%$)

p	Z^*	Z_H	# Circles	CPU Time (secs)
2	85.566	85.566	38583	23.67
5	49.649	49.649	14467	7.98
10	29.921	29.921	3563	0.60
15	21.319	24.910	2444	0.60
20	16.101	21.319	1830	0.54
25	13.210	16.867	1371	0.42
30	11.161	14.603	1148	0.63
Average	**32.418**	**35.076**	**9058**	**4.92**

Table 3.3: Results for the Duality Gap Investigation for $MSEHA$ ($DV = 0\%$)

Results show that the tighter the upper bound, the faster the optimal solution is found. This is because the tighter the upper bound, the more circles can be discarded due to having radii larger than the upper bound. This creates less possible facilities locations and so makes the problem smaller and relatively easier to solve. In other words, CPLEX can find the optimal solution relatively much quicker as many calculations are

eliminated.

Manipulating the solution value obtained using CPLEX based on the size of the duality gap is revisited in the development of an exact algorithm in Chapter 4, where a duality gap policy is established in order to refine the algorithm so that it can optimally solve large continuous $p-$centre problems.

We shall now discuss a new scheme devised to aid in the generation of potential facility locations.

3.3.3 The Corridor Method

The aim of this method is to save computational time when finding all potential facility locations generated from the circles formed from three demand points (i.e. three critical points). This idea was developed so that we did not need to calculate all the angles for each combination of three demand points to determine whether or not they form an acute triangle (i.e. the 'angle method' proposed by Chen & Handler (1987), see Section 2.2.4, Chapter 2). As the new scheme systematically incorporates the same circles as the angle method, it can be considered as a reduction method without eliminating any of the possible solutions.

The Main Idea

Take any two points in the plane and draw the unique straight line, l_1, that passes through them both. The two lines, perpendicular to l_1, that pass through the centre of both points are then calculated. The area in between these two lines is the matter of interest. Any points that lie on the line or outside of this area can be discarded for now, as these points will form a right-angled or an obtuse triangle with the given two points. The only (but not all) points that make acute triangles will lie inside this region.

Figure 3.3 shows the 'corridor' for points P_1 and P_2 (lighter area labeled 'corridor') and the area where the points are not checked for further calculations (darker area labeled 'reject'). Figure 3.3 shows that the one place inside the corridor that allows points to form a right-angled or obtuse triangle with P_1 and P_2 is inside the circle defined by P_1 and P_2. We wish to find all the points that lie inside the lighter area or

the corridor. That is, the points that lie inside the corridor but outside of the circle.

As shown in Figure 3.3, there are two cases where points can be rejected. This includes (i) if the points lies outside of the corridor region and (ii) if the points lie inside the circular region.

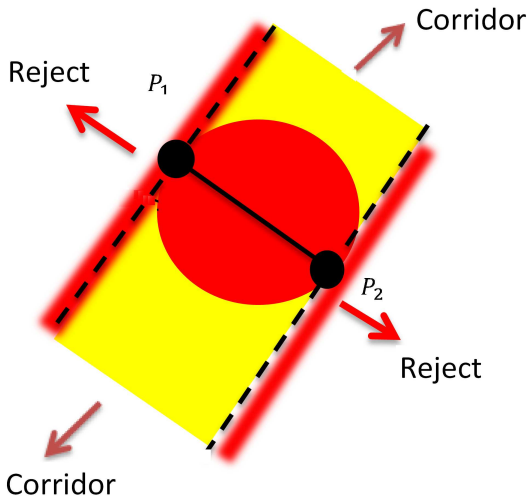


Figure 3.3: The corridor for points P_1 and P_2.

It is important to note that a similar diagram can be seen in Elzinga & Hearn (1972: 383). Even though the purpose of this diagram is demonstrating where the next point can be chosen for the EHA, the mathematics are the same as Elzinga & Hearn are trying to find any point in the plane that creates an acute triangle with two given points. Therefore, the corridor method extends the work devised by Elzinga & Hearn by using their idea to find all third points that create acute triangles with any two given points.

Corridor Method Mathematically Analysed

Let G be a $(n_1 \times n_2)$ grid. Take two points $P_{k,l}$ and $P_{s,t}$ respectively, such that:

$$k = (1, 2, \ldots, (n_1 \times n_2 - 1)),$$

$$l = (1, 2, \ldots, (n_1 \times n_2 - 1)),$$

$$s = ((k+1), \ldots, (n_1 \times n_2)),$$

$$t = ((l+1), \ldots, (n_1 \times n_2)).$$

The equation of the line that joins the two points $P_{k,l}$ and $P_{s,t}$ is

$$y = \frac{t-l}{s-k}(x-k) + l, \tag{3.23}$$

and the two lines of equation that make up the corridor are

$$y = \frac{k-s}{l-t}(x-k) + l, \tag{3.24}$$

and

$$y = \frac{k-s}{l-t}(x-s) + t. \tag{3.25}$$

Case 1: Inside the corridor

Take any point $P_{x,y}$ in the plane. If it does not satisfy Equation (3.24) and (3.25), that is

$$k < x < s, \tag{3.26}$$

and

$$\frac{k-s}{l-t}(x-k) + l < y < \frac{k-s}{l-t}(x-s) + t, \tag{3.27}$$

then it does not lie inside the corridor region for points $P_{k,l}$ and $P_{s,t}$ and can therefore be rejected.

If, however, the point $P_{x,y}$ does lie inside the corridor region for points $P_{k,l}$ and $P_{s,t}$, case 2 must now be checked.

Case 2: Outside of the circle

If the point lies inside the corridor region, certain checks must be performed to see if it lies in the circular region or not. We wish to keep all points that lie outside the circle region, as they will form an acute triangle with points $P_{k,l}$ and $P_{s,t}$. That is

$$y > \sqrt{r^2 - (x - x_c)^2} + y_c, \tag{3.28}$$

where x_c and y_c are the centre points of the line $P_{k,l} \rightarrow P_{s,t}$, and $r = \frac{dist_{P_{k,l},P_{s,t}}}{2}$ where $dist_{P_{k,l},P_{s,t}}$ is the Euclidean distance from point $P_{k,l}$ to $P_{s,t}$.

This is achieved as follows:

a) Calculate the radius, r, and centre, (x_c, y_c), of the circle formed by points $P_{k,l}$ and $P_{s,t}$.

b) Calculate the Euclidean distance $dist_{P_{x,y},P_{x_c,y_c}}$. If $dist_{P_{x,y},P_{x_c,y_c}} > r$, we save the point $P_{x,y}$ for further investigation, otherwise it is rejected.

Once we have found all the points that lie outside of the circular region, we can eliminate these points from the search to obtain the set of saved points that form an acute triangle with $P_{k,l}$ and $P_{s,t}$.

Each saved point is added to the set

$$Sav(P_{k,l}, P_{s,t}) = \{P_{x,y} \mid x \text{ satifies (3.26) and } y \text{ satisfies (3.27) and (3.28)}\},$$

and the total number of saved points S is

$$S = \sum_{k,l=1}^{n_1 \times n_2 - 1} \sum_{s=k+1}^{n_1 \times n_2} \sum_{t=l+1}^{n_1 \times n_2} |Sav(P_{k,l}, P_{s,t})|. \tag{3.29}$$

Investigative Results

The corridor method was tested against Chen and Handler's (1987) well-known angle method. Two data sets from the TSP-library, namely $pr439$ and $rat575$, were tested for comparison and the results can be seen in Tables 3.4 and 3.5.

The tables are structured in the following way. The first column shows the number of facilities we wish to locate. The second column, labelled Z_H, shows the upper bound generated from the H_2 heuristic proposed by Drezner (1984a) (this heuristic will be discussed further in Chapter 4). The third column shows the number of circles (made from three demand points) that was found for both the angle and corridor method. The fourth and fifth columns show the computational time in seconds each method required to find that number of circles. Finally, the last column shows the percentage decrease in computational time using the corridor method compared to the angle method.

			CPU Time (secs)		
p	Z_H	# Circles	Angle Method	Corridor Method	% Decrease
10	1716.510	587409	17.67	1.93	89.05
20	1169.540	292182	17.66	1.97	88.87
30	975.00	191794	17.61	2.00	88.66
40	874.271	142327	17.41	1.97	88.71
50	580.005	39258	17.24	1.98	88.50
60	570.088	36985	17.35	2.01	88.40
70	503.271	24565	17.99	2.1	88.20
80	467.039	18698	17.46	2.05	88.29
90	391.511	10007	17.14	2.17	87.35
100	315.486	4138	17.04	1.97	88.47
Average	**756.27**	**134736**	**17.46**	**2.02**	**88.45**

Table 3.4: Angle vs. Corridor ($n = 439$)

			CPU Time (secs)		
p	Z_H	# Circles	Angle Method	Corridor Method	% Decrease
10	69.426	832370	39.30	3.39	91.39
20	48.107	241223	40.19	3.45	91.42
30	39.655	120212	41.10	3.42	91.68
40	33.365	63518	39.90	3.41	91.46
50	30.336	44849	38.60	3.37	91.26
60	27.951	32710	38.44	3.37	91.23
70	25.578	22952	38.34	3.47	90.94
80	24.135	18219	38.06	3.38	91.12
90	21.932	12597	38.27	3.52	90.80
100	20.402	9419	38.67	3.53	90.87
Average	**34.090**	**139807**	**39.09**	**3.43**	**91.22**

Table 3.5: Angle vs. Corridor ($n = 575$)

The results are very promising, with an average decrease in computational time of over 88.4% and 91.2% for $n = 439$ and $n = 575$ respectively. Although the angle method is still quick, this decrease in computational time could make a significant difference for algorithms where the potential facility locations are constantly re-calculated. Thus, the corridor method shows much value, and therefore it is incorporated into all further algorithms that are developed during this research.

3.4 Summary

This chapter has studied the relevant formulations used in this research, namely the Set Covering problem, the vertex $p-$centre problem and the continuous $p-$centre problem.

It has investigated the usefulness of combining heuristic and optimal methods together to obtain the optimal solution. The effect on the solution value quality when the duality gap value is manipulated was also explored. Finally, a new method that found all potential facility locations formed from three demand points was discussed and proven to require less computational time in comparison a well known and popular method.

The next chapter will revisit a dormant, but interesting, optimal algorithm.

Chapter 4

An Enhanced Implementation of Drezner's Exact Method

4.1 Introduction

This chapter begins by testing the original implementation of Drezner's optimal algorithm for the $p-$centre problem to highlight its strengths and weaknesses. New enhancements, designed to speed up the algorithm by avoiding unnecessary calculations, are then proposed and mathematically supported with lemmas and proofs. Furthermore, a duality gap policy is also devised to improve the algorithm's efficiency for data sets with more complex distributions that are harder to optimally solve. The enhanced algorithm is then evaluated under two scenarios. Scenario One finds the optimal solution for five TSP-Library data sets, namely $pr439$ (a 439-city problem), $rat575$ (a 575-rattled grid problem), $rat783$ (a 783-rattled grid problem) and $pr1002$ and $rl1323$ (a 1002-city and a 1323-city problem respectively). The algorithm demonstrates its robustness as optimal solutions are found for the first time for the data sets where $n \geq 575$. Scenario Two tests the revised algorithm on nine generated data sets where n ranges from $400-800$ with distinguished and contrasting distributions, namely randomly spread, semi-clustered and clustered, to emphasis the algorithm's strength.

4.2 $Z-$maximal Circles

4.2.1 Introduction and Definitions

Drezner proposed an exact method to find the optimal solution for the $p-$centre problem in the plane using a group of circles called $Z-$*maximal circles*. A circle is defined as maximal based on a given upper bound, Z. The set of $Z-$maximal circles are identified and their centres are used as a subset of potential facility locations to find the optimal solution value. An initial upper bound is found using a heuristic method,

and each iteration of the algorithm either finds a better solution (i.e. a smaller upper bound) which becomes the new upper bound, or proves that there is none and so has reached optimality.

Let us begin by defining the additional notations and concepts that are needed.

Definition 4.2.1. *The* **closure** *of circle C_j is the set of demand points encompassed by circle C_j which is defined as*

$$Cl_j = \{i \ \in I|\ d_{i,j} \leq r_j\} \ \forall\ j = 1 \ldots m.$$

Let:
K: a subset of I;
$R(K)$: radius of the smallest circle encompassing all points in K;
J_Z: set of $Z-$maximal circles ($J_Z \subset J$);
$d'_{i,l}$: Euclidean distance from demand point i to demand point l;
Z: the upper bound at a given iteration.

Definition 4.2.2. *The* **minimum covering circle** (MCC) *of the set K is the smallest circle encompassing all points in K with radius $R(K)$.*

We can now define a $Z-$maximal circle. The following definition was taken directly from Drezner (1984a).

Definition 4.2.3. *A circle C_j with radius r_j is said to be $Z-$***maximal** *(often simply called maximal) if:*

1. *$r_j < Z$;*

2. *For every demand point $i \notin Cl_j$, $R(Cl_j \cup \{i\}) \geq Z$.*

In other words, if $r_j \geq Z$, then circle C_j cannot be classified as a $Z-$maximal circle. However, if $r_j < Z$, the next step is to add a demand point $i \notin Cl_j$ and find $R(Cl_j \cup \{i\})$. If $R(Cl_j \cup \{i\}) \geq Z \ \ \forall\ i \ \notin Cl_j$, then circle C_j is said to be $Z-$maximal.

4.2.2 Drezner's Optimal Algorithm

Introduction

Drezner proposed two ways to solve the $p-$centre problem using $Z-$maximal circles.

The first, which will be referred to as $For_0^{(a)}$, uses the set covering problem (formulated in Chapter 3) to find the minimum number of $Z-$maximal circles needed. For clarity, we shall define $For_0^{(a)}$ below.

$$(For_0^{(a)}) : \text{Minimise} \quad \sum_{j \in J_Z} x_j \tag{4.1}$$

$$\text{subject to} \quad \sum_{j \in J_Z} A_{i,j} x_j \geq 1 \quad \forall i \in I, \tag{4.2}$$

$$x_j \in \{0,1\} \quad \forall j \in J_Z, \tag{4.3}$$

$$\text{where} \quad A_{i,j} \begin{cases} 1 \text{ if } i \in Cl_j, \\ 0 \text{ else,} \end{cases} \tag{4.4}$$

$$x_j = \begin{cases} 1 \text{ if } Z-\text{maximal circle } C_j \text{ is selected,} \\ 0 \text{ else.} \end{cases} \tag{4.5}$$

The objective function (4.1) refers to minimising the number of $Z-$maximal circles required to cover the set of demand points. Constraint (4.2) guarantees that every demand point is encompassed, or covered, by at least one $Z-$maximal circle. If the minimum number found in (4.1) is $\leq p$, then the upper bound is decreased by setting Z to the radius of the largest $Z-$maximal circle from the obtained solution. The process of identifying the $Z-$maximal circles is then repeated. Else (i.e. the minimum number is $> p$), the current upper bound Z is taken as the optimal solution and the algorithm terminates.

In the second method, referred to as $For_0^{(b)}$, a new constraint (4.6) is added to $For_0^{(a)}$ to impose that the number of covering circles has to be equal to p, while the objective function (4.1) is omitted turning the problem into a feasibility problem.

$(For_0^{(b)})$: Find $x_j \in \{0,1\}$ $\forall j \in J_Z$ subject to

(4.2) - (4.3),

$$\sum_{j \in J_Z} x_j = p. \tag{4.6}$$

Before we use Drezner's optimal algorithm to solve the $p-$centre problem using $Z-$maximal

circles as described in Figure 4.1, we shall first define the following additional notations.

C_J^1: the set of null circles created from one critical point only (i.e., $r_j = 0 \; \forall \; C_j \in C_J^1$);
C_J^2: the set of circles created from two critical points defining its diameter;
C_J^3: the set of circles created from three critical points forming an acute triangle.

Step 1. Find the three sets of circles C_J^1, C_J^2 and C_J^3.

Step 2. Find an initial solution and set the solution value as the initial upper bound, Z.

Step 3. Eliminate all circles whose radii are $\geq Z$ from C_J^2 and C_J^3.

Step 4. Find all $Z-$maximal circles using Procedure $FMC(C_J^1, C_J^2, C_J^3, Z, J_Z)$ (see Figure 4.2).

Step 5. Solve $CP_0^{(a)}$ or $CP_0^{(b)}$ using the $Z-$maximal circles J_Z.

If the optimal value of $CP_0^{(a)}$ is $\leq p$ or $CP_0^{(b)}$ is feasible, set the new upper bound Z as the radius of the largest $Z-$maximal circle found in the solution and go to Step 3.

Else take the upper bound Z as the optimal solution value of the planar $p-$centre problem and stop.

Figure 4.1: Drezner's Original Algorithm (Drezner (1984a))

It is important to note that an appropriate heuristic must be used to find an initial upper bound. For instance, a simple multi start heuristic can be used. In this study we opted for the H_2 heuristic proposed by Drezner (1984a) for consistency reasons.

Computing all $Z-$maximal Circles

We now construct an algorithm that finds which circles (i.e. the set of potential facility locations) are $Z-$maximal. This 'finding $Z-$maximal circles' algorithm will be referred to as the FMC algorithm for short and is given in Figure 4.2.

Time Complexity

The time complexity of the algorithm is examined at each of the steps described in Figure 4.1. The complexity is bounded by Step 5 which is NP-hard. However, let us examine the other four steps.

Procedure $FMC(C_J^1, C_J^2, C_J^3, Z, J_Z)$

Step 1. Set $J_Z = \emptyset$.

Step 2. (Find all $Z-$maximal circles in the set C_J^1.)

For each $C_j = \{i\} \in C_J^1$ do: If $\frac{d'_{i,l}}{2} > Z \ \forall \ l \neq i \in I$, add C_j to J_Z.
Else the (single point) circle C_j is not $Z-$maximal.

Step 3. (Find all $Z-$maximal circles in the set $C_J^2 \cup C_J^3$.)

For all $C_j \in \{C_J^2 \cup C_J^3\}$ do:
If $R(Cl_j \cup \{i\}) > Z \ \forall \ i \notin Cl_j$ (i.e. the circle C_j is $Z-$maximal by definition) set $J_Z = J_Z \cup \{C_j\}$.
Else, the circle C_j is not $Z-$maximal.

Step 4. Return J_Z.

Figure 4.2: The FMC Algorithm

In Step 1, the total number of circles created from one, two or three demand points is at most $n_0 = \frac{n(n^2+5)}{6}$, giving a time complexity of $O(n^3)$. If the H_2 heuristic is used in Step 2, it has a time complexity of $O(n^2)$ which is dominantly the Elzinga-Hearn algorithm (see Drezner & Shelah (1987)). Note that H_2 is run h times and the best is chosen. The wost-case time complexity for Step 2 is $O(n^3)$, and the worst case for Step 3 has a complexity of $O(n)$. Finally, the worse case scenario in Step 4 would be to calculate all the $Z-$maximal circles given so that every possible circle is evaluated. If the closure of circle C_j holds n_j demand points, the time complexity for this step becomes $O(n^6)$ as this requires $\sum_{j=1}^{n_0}(n - n_j)n^2$ calculations.

4.3 Initial Results

All algorithms in this study were coded in $C++$ on a HP Elitebook 8570w with 12GB of memory. The IBM ILOG CPLEX 12.6 console was incorporated into the program using default parameters. Note that in the basic trial, squared distances were also adopted when useful to improve code efficiency (e.g. when distances are compared, or for non-acute triangle detection).

The algorithm used to find the MCC is the algorithm of Elzinga & Hearn (1972), though a more advanced one could be used (see Elshaikh *et al* (2015)).

4.3.1 Results using Drezner's suggested formulations

Initial results were found for the TSP-Library data set $pr439$ with used the two formulations described by Drezner, $For_0^{(a)}$, $For_0^{(b)}$, to solve the 90−centre problem. For the data set $pr439$, the problem was optimally solved using $For_0^{(a)}$ requiring more than 38 hours (i.e. 137692.6 seconds) and using 4580 iterations. When using $For_0^{(b)}$ for the same data set, the required computational time was reduced to just below 3 hours (10654.30 seconds) and using 393 iterations. For $rat575$, an optimal result was obtained using $For_0^{(b)}$, however it required nearly 30 hours (107916.0 seconds) of computational time using 2729 iterations. Furthermore, when using $For_0^{(a)}$, the program was stopped after the time limit of 2 days with only a feasible solution found with a solution value of 21.471 (a percentage difference of 18% from the optimal solution). It will be shown later that the optimal solution can be found in less than half an hour (996.43 seconds) with our improved method. Therefore, this example highlights the importance of developing ways to enhance the efficiency of Drezner's optimal algorithm. Our first attempt is to examine the formulation of the $p-$centre problem, which will be discussed in the next section.

4.3.2 Modification of the Covering Problem (Enhancement Zero)

The formulation for the $p-$centre problem (For_{pc} given initially in Chapter 3) was the first to be examined. It is given below for completeness.

$$(For_{pc}) \text{ Minimise} \qquad W \tag{4.7}$$

$$\text{subject to} \qquad \sum_{j \in J_Z} A_{i,j} x_j \geq 1 \quad \forall i \in I, \tag{4.8}$$

$$\sum_{j \in J_Z} x_j = p, \tag{4.9}$$

$$x_j r_j \leq W \quad \forall j \in J_Z, \tag{4.10}$$

$$x_j \in \{0,1\} \ \forall j \in J_Z. \tag{4.11}$$

where

W: the maximum distance between a facility and a demand point.

This formulation For_{pc} has an advantage over Drezner's original two suggestions. This is due to the optimal solution value being established quicker as there will be a smaller amount of iterations. Although For_{pc} may be harder to solve than the set covering problem, the advantages meant that it was the chosen formulation for this investigation. Therefore, the introduction of For_{pc}, instead of $For_0^{(a)}$ or $For_0^{(b)}$, could be considered as our first enhancement due to generating tighter bounds. However, for simplicity and conciseness, the results of For_{pc} will be used as a starting point from which we will base our improvements. In other words, whenever we refer to Drezner's original algorithm we mean the algorithm with For_{pc} instead.

4.3.3 Results using For_{pc}

Tables 4.1 and 4.2 show the optimal results for $p = 10$ to 100 in multiples of 10. The first column, titled p, shows the number of facilities required. The second column shows the initial upper bound value, denoted by Z_1, that was found from a 1000 iteration runs of the H_2 heuristic described in Drezner (1984a). The computational time to find this upper bound is also given in the corresponding CPU Time column. The fourth column, titled Z^*, shows the optimal solution found from Drezner's optimal algorithm, followed by the loop computational time consumed by Drezner's optimal algorithm only (i.e. excluding the computational time required for the H_2 heuristic).

Other information, such as the number of loops (iterations) before an optimal solution value was found, the total time spent on computing the $Z-$maximal circles and the total time spent on solving the problem in CPLEX are reported. The percentages of these times are also given in the remaining columns to indicate proportionally how the overall computational time was shared over the different calculations. These will provide us with an insight into the problem and hence potential ways on how to speed the search up. Note that these two individual percentages when added are below 100% due to other calculations that have not been included such as the initial construction of circles (Step 1 in Figure 4.1).

These results were first reported in Elshaikh *et al* (2016) for evaluating a heuristic known as the perturbation method.

	H_2 Heuristic		Optimal Solution						
p	Z_1	CPU Time	Z^*	Loop CPU Time (secs) [a]	# Loops	Maxi Circles (secs)	CPLEX (secs)	Maxi Circles (%)	CPLEX (%)
10	1716.510	96.88	1716.510	6252.72	2	6154.93	36.39	98.44	0.58
20	1169.540	170.28	1029.710	56753.00	36	54203.60	297.90	95.51	0.52
30	975.000	205.36	739.193	37017.10	49	35024.50	222.96	94.62	0.60
40	874.271	218.90	580.005	31355.00	67	29986.40	209.61	95.64	0.67
50	580.005	235.61	468.542	4939.25	38	4781.67	59.91	96.81	1.21
60	570.088	246.86	400.195	4956.45	47	4794.88	57.77	96.74	1.17
70	503.271	256.30	357.946	3170.89	46	3076.31	39.04	97.02	1.23
80	467.039	300.01	312.500	2186.27	53	2109.08	37.33	96.47	1.71
90	391.511	276.20	280.903	1258.22	48	1214.45	23.80	96.52	1.89
100	315.486	332.53	256.680	462.30	32	437.38	13.93	94.61	3.01
Average	**756.272**	**233.89**	**614.218**	**14835.12**	**42**	**14178.32**	**99.87**	**96.24**	**1.26**

[a] This excludes computational time for the H_2 heuristic.

Table 4.1: Initial Results for $n = 439$ TSP-Lib

	H_2 Heuristic		Optimal Solution						
p	Z_1	CPU Time	Z^*	Loop CPU Time (secs) [a]	# Loops	Maxi Circles (secs)	CPLEX (secs)	Maxi Circles (%)	CPLEX (%)
10	69.426	98.34	67.926	83898.60	10	78805.90	351.59	93.93	0.42
20	48.107	175.62	45.475	19087.06	11	17513.80	519.37	91.75	2.72
30	39.655	238.26	35.556	9743.91	14	8698.37	577.51	89.27	5.93
40	33.365	296.90	30.063	41733.00	11	3240.15	38342.30	7.76	91.88
50	30.336	403.76	25.826	9612.61	15	2515.16	6985.51	26.17	72.67
60	27.951	422.18	23.163	28344.00	18	1938.64	26327.70	6.84	92.89
70	25.578	558.86	20.858	40256.90	20	1449.39	38756.30	3.60	96.27
80	24.135	535.90	19.026	40181.70	17	892.371	39247.90	2.22	97.68
90	21.932	743.20	17.460	4260.10	18	696.769	3532.50	16.36	82.92
100	20.402	795.13	16.420	33694.00	15	405.90	33262.20	1.20	98.72
Average	**34.089**	**426.81**	**30.177**	**31081.24**	**15**	**11615.65**	**18790.29**	**33.91**	**64.21**

[a] This excludes computational time for the H_2 heuristic.

Table 4.2: Initial Results for $n = 575$ TSP-Lib

It can be noted for the data set $pr439$ that when $p = 90$ and 100 the computational times are 1258.22 and 462.30 seconds respectively. This is therefore much quicker than using the fastest of either $For_0^{(a)}$ or $For_0^{(b)}$ which consume 10654.30 and 2989.64 seconds respectively. Therefore, the use of the formulation For_{pc} is approximately 9 and 7 times faster than using $For_0^{(a)}$ or $For_0^{(b)}$ in these two instances.

4.3.4 Interesting Observations

The strengths and weaknesses of this optimal approach to solve medium to large continuous $p-$centre problems can be seen from the results. An obvious strength to highlight is that optimal solution values for the $rat575$ data set have now been discovered for all p values.

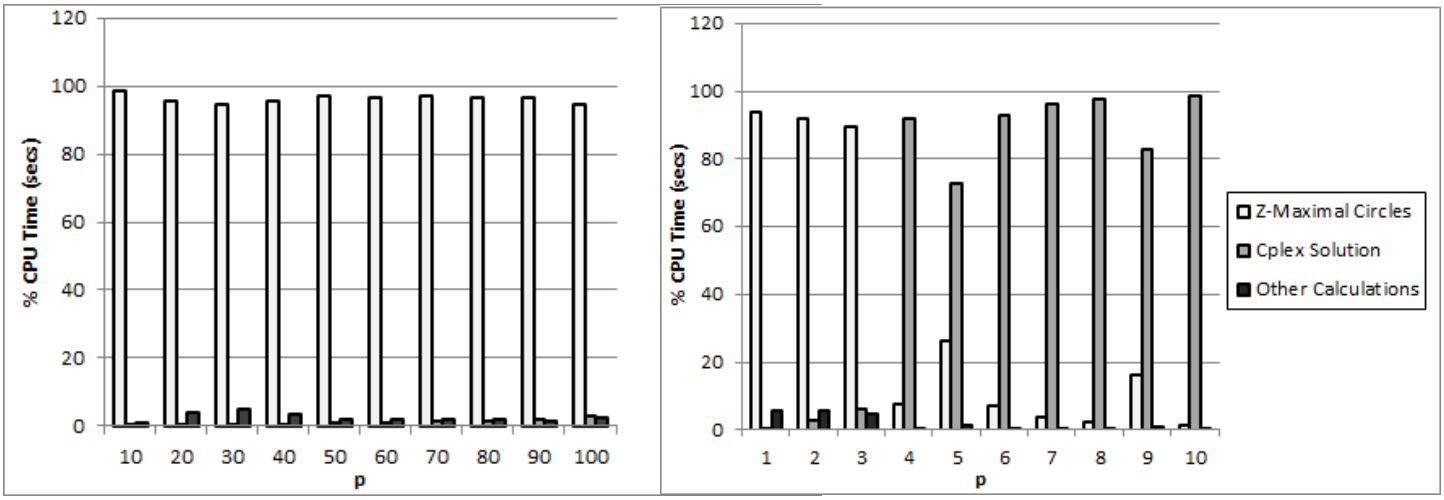
However, by studying Tables 4.1 & 4.2, it can be observed that the time shared between the determination of the $Z-$maximal circles and CPLEX differ in these two instances and for the value of p. There are two areas where enhancements could be introduced in an attempt to shorten the overall computational time.

a) the way the $Z-$maximal circles are identified from one iteration to the next;

b) a choice of a compromise between the quality of a feasible solution and its corresponding computational time when solving For_{pc} (i.e. finding an optimal solution or a good feasible solution).

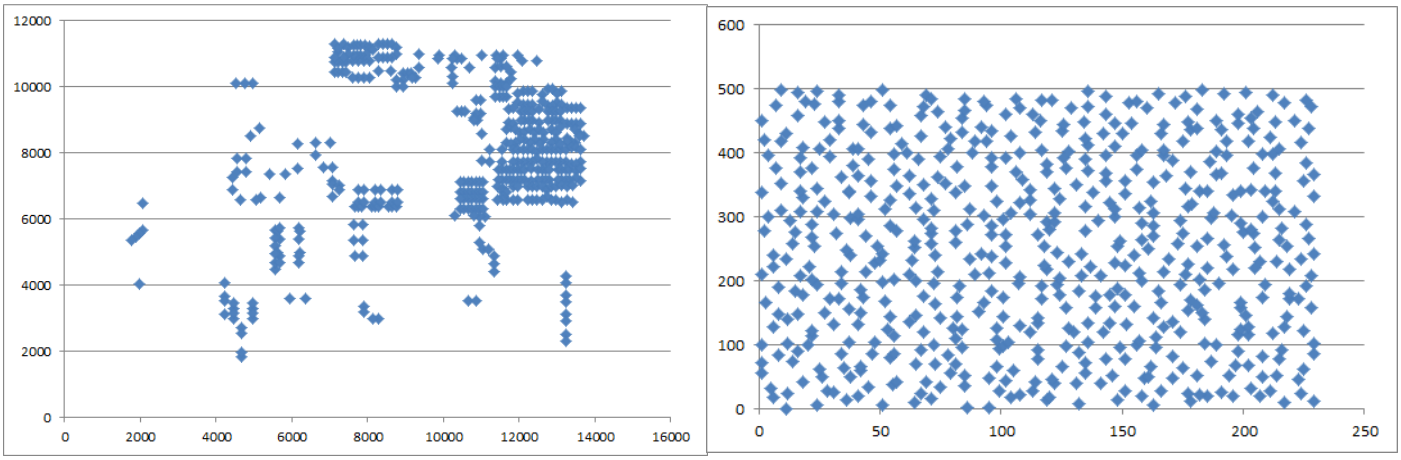
It is interesting to observe that the majority of the computational time for the data set $pr439$ was taken by calculating the $Z-$maximal circles, whereas the majority of computational time was consumed when solving For_{pc} in CPLEX for $rat575$. This is demonstrated in Figure 4.3. This could be due to the distribution of the two data sets, which can be visually compared using Figure 4.4.

Figure 4.4a shows that $pr439$ has a clustered distribution. This type of distribution may affect computational time when classifying circles as $Z-$maximal or not. If a circle is small or lies in a sparse area of demand points, the number of demand points not enclosed within the circle may be proportionally high. Therefore, determining whether the circle is $Z-$maximal or not may require many combinations and calculations. However, as the covering circles vary in size it is more likely that the cardinality



(a) *pr*439 (b) *rat*575

Figure 4.3: Comparing time spent to calculate $Z-$maximal circles, the cplex solution and other calculations



(a) *pr*439 (b) *rat*575

Figure 4.4: Distribution for *pr*439 and *rat*575 from TSP-Lib

of set of $Z-$maximal circles will be smaller in this instance compared to a data set with an evenly spread distribution. Therefore, the calculation time is mainly taken up by calculating $Z-$maximal circles rather than finding a solution in CPLEX.

There is a marked difference in the distribution for the data set where $rat575$, as the demand points are evenly spread. The results for this data set show an interesting pattern. When p is small $(10-30)$ the computational times are proportionally similar to that of the data set $n = 439$. However, as p increases, the time spent finding a solution in CPLEX dramatically increases, and time spent computing $Z-$maximal circles greatly decreases proportionally. This could be explained by looking at the distribution. The smaller p is, the larger the upper bound value will be. This means there will be more circles that are accepted by condition (1) in Definition 4.2.3. Thus there are more circles to check condition (2) with, and this takes much computational time. However, the size of the $Z-$maximal circles set will be relatively small and so finding an optimal solution will not take as much computational time (much like the reasoning for the $pr439$ data set). As p gets larger, the number of circles that satisfy condition (1) in Definition 4.2.3 decreases, and so finding the $Z-$maximal circle set does not take as much computational time. However, if a circle is classified as $Z-$maximal, then it is likely that there are other circles of a similar size that will also be classed as $Z-$maximal, as the evenly spread data can given a 'duplicate effect' but creating many circles of similar sizes. Furthermore, as the distribution is evenly spread it could mean that most circles that satisfy condition (1) are $Z-$maximal. This could therefore lead to a large subset of potential facility locations and so increasing the time spent on finding a solution.

4.4 The $Z-$Maximal Circles-Based Enhancements

If the algorithm only needed a small number of iterations, the original proposal from Drezner would be efficient and therefore enhancements might not be worthwhile. However, as the algorithm involves many iterations, it soon becomes apparent that information found in one iteration could be used to save computational time in the next. As the cumulative computational time saving from the improvements might create a big change to the overall computational time, it is worth investigating and testing possible

enhancements. This is especially useful for larger data sets, as some of the previous results show the algorithm can take well over 24 hours to complete.

4.4.1 Enhancement One: EHA-Based Implementation

As described in Chapter 1, the EHA is used to find the solution to the single facility location problem, and thus find $MCCs$. As this is repeatedly needed in Step 3 of the FMC algorithm, the implementation of this algorithm was enhanced in two ways in an attempt to reduce the overall computational time which will now be explained.

Early Termination

The EHA starts with a circle made from any two selected points and continues to find a covering circle of increasing size until all points are covered. Therefore, EHA terminates when the MCC is found. However, it is important to realise that the exact centre point and the radius of the MCC are not needed for this maximal circles-based method: we simply aim to establish whether or not the MCC will be larger or smaller than the upper bound Z. With this in mind, we can see that if the radius of the covering circle found at any point during the EHA is smaller than the upper bound, then the EHA will have to continue to the end as normal. However, it can be terminated early if the covering circle's radius ever exceeds Z. This is because at each iteration in the EHA, the new circle's radius is either the same or larger. Therefore, if a circle has a radius $\geq Z$ at any point, there is no need to continue as the final circle (MCC) will be either the same or even larger. As this is all the information needed, the algorithm can terminate early and the next combination can be tested. Thus, not having to necessarily compute the radius of the MCC each time will save computational time.

More Informative Initial Points

Instead of starting the EHA from random points or selecting points using selection rules, such as the ones adapted by other authors including Welzl (1991) and Elshaikh *et al* (2015), here we consider the information we have already found. In other words, the two or the three critical points that defined the circle being evaluated are our chosen initial points for the EHA. This makes the selection deterministic and yields faster results.

Combining these two ideas forms a double enhancement, which will be referred to as Enh1. Enh1 was tested on the data set $pr439$ where $p = 10, \ldots, 100$. The results are displayed in Table 4.3, where the fifth column shows the computational time required for Drezner's optimal method, and the sixth column gives the computational time needed for Drezner's optimal method where Enh1 is incorporated. The last column, displaying the overall percentage decrease in computational time, shows a significant drop in required time with an average decrease of over 58%. For clarity, the computational time required by the H_2 heuristic is given in the second column.

	H_2 Heuristic	Drezner's Optimal Method				
p	CPU Time (secs)	Z^*	# Loops	Original CPU Time (secs)[a]	New CPU Time (secs)[a]	% Overall Change
10	96.88	1716.510	2	6252.72	1949.32	68.83
20	170.28	1029.710	36	56753.00	19323.00	65.95
30	205.36	739.193	49	37017.10	12978.50	64.94
40	218.90	580.005	67	31355.00	11434.00	63.53
50	235.61	468.542	38	4939.25	2019.48	59.11
60	246.86	400.195	47	4956.45	2082.34	57.99
70	256.30	357.946	46	3170.89	1368.08	56.86
80	300.01	312.500	53	2186.27	1024.88	53.12
90	276.20	280.903	48	1258.22	654.80	47.96
100	332.53	256.680	32	462.30	264.04	42.88
Average	**233.90**	**302.010**	**45**	**14835.12**	**5309.84**	**58.12**

[a] This excludes computational time for the H_2 heuristic.

Table 4.3: Comparing CPU Times (in secs) for Drezner's optimal method with and without Enh1 ($n = 439$)

Note that Enh1 does not affect the total number of iterations of the algorithm used to determine the solution, and it will incorporated into Step 3 of the FMC algorithm only.

Summary

- If at any point in the Elzinga-Hearn algorithm a circle is created whose radius $\geq Z$, the algorithm can be terminated early.
- Instead of starting the algorithm on random demand points, it starts with the critical points of circle C_j.

4.4.2 Enhancement Two: Efficiently Recording $Z-$maximal Circles

At each new iteration in the Drezner algorithm, the process of finding the $Z-$maximal circles begins again from the start irrespective of earlier iterations. However, when examining the first set of results, we observed that it was likely that many circles were being classified as $Z-$maximal for successive iterations.

As an example, Table 4.4 displays the number of $Z-$maximal circles found for the first 10 iterations of the original algorithm using the data set $pr439$ where $p = 100$. The first column shows the iteration number, and the second column gives the total number of circles found after completing Steps 1−3 in Figure 4.1. The third column shows the number of circles that are classified as maximal during this iteration, and the fourth column gives the number of maximal circles that were already classified as maximal from previous iterations. Finally, the last column displays the percentage of new circles classified as maximal in the given iteration.

Iteration #	# Original Circles	# $Z-$maximal Circles	# Circles Previously Identified	Extra % Required
1	9281	860	-	-
2	9189	855	780	8.77
3	8835	797	597	25.09
4	8796	805	758	5.84
5	8652	809	684	15.45
6	8449	798	640	19.80
7	8384	804	735	8.58
8	7922	756	478	36.77
9	7855	767	693	9.64
10	7637	770	601	21.95
Average	**8500.00**	**802.10**	**662.88**	**16.88**

Table 4.4: Number of $Z-$maximal circles required & previously identified for the first 10 iterations ($n = 439, p = 100$)

When studying the fourth column, in this example we can see the vast majority of classified $Z-$maximal circles have been previously found, with only a small percentage of new circles classified as $Z-$maximal. An average of only 16.88% extra $Z-$maximal circles to be added is required (note that the first iteration was not included), with the largest deviation being 36.77%. Therefore, it is worthwhile to track which circles have been defined as $Z-$maximal and which have not to avoid unnecessary calculations.

A technique to identify whether a circle is $Z-$maximal or not can now be constructed. For clarity, let K be a subset of I.

Lemma 1. *If circle C_j is Z_t-maximal for an upper bound Z_t at iteration t, then it is also $Z_{t+1}-$maximal for iteration $t+1$ if and only if its radius $r_j < Z_{t+1}$.*

Proof. We know at each iteration t, the upper bound Z strictly decreases. Therefore, we can say $Z_t > Z_{t+1}$. For circle C_j to be a $Z-$maximal circle at iteration t, the following two conditions need to be satisfied:

1. $r_j < Z_t$;
2. For every demand point $i \in I$ such that $i \notin Cl_j,\ R(Cl_j \cup \{i\}) \geq Z_t$.

As $Z_{t+1} < Z_t$, we can deduce that $R(Cl_j \cup \{i\}) > Z_{t+1}$. Thus if $r_j < Z_{t+1}$, circle C_j will still be a $Z-$maximal circle by definition at iteration $t+1$. □

The information denoting whether or not circle C_j has been found to be $Z-$maximal or not can be stored in a binary or logical vector $CircMax$ where

$$CircMax_j = \begin{cases} 1 \text{ if } C_j \in J_{Z_t}, \\ 0 \text{ else.} \end{cases}$$

In brief, if circle $C_j \in J_{Z_t}$, then $C_j \in J_{Z_{t^*}}$ where $t^* > t$ and $r_j < Z_{t^*}$, and so there is no need for further calculations in subsequent iterations.

Lemma 1 is incorporated into Steps 2 and 3 of the FMC algorithm to avoid performing redundant calculations. We will refer to this enhancement as Enh2.

Summary

If circle $C_j \in J_{Z_t}$, then $C_j \in J_Z$ for all subsequent iterations and can be disregarded from further calculations.

4.4.3 Enhancement Three: Fast Identification of some Non$-Z-$maximal Circles

This enhancement, which we will refer to as Enh3, uses a technique to quickly find some non$-Z-$maximal circles without performing unnecessary calculations.

Take the circle C_j with a centre point (x_{j_c}, y_{j_c}) and radius r_j. We can now create a new circle C_j^+ centered at (x_{j_c}, y_{j_c}) with radius Z. It is clear that C_j lies inside C_j^+.

Lemma 2. *If $s \in I$ is not covered by C_j (i.e. $s \notin Cl_j$) but is strictly covered by C_j^+, then circle C_j is not $Z-$maximal.*

Proof. Let $s \in I$ with $s \notin Cl_j$ but strictly covered by C_j^+. Then the smallest circle, C, containing s and the whole circle C_j, contains all the points in Cl_j and is strictly contained in C_j^+. Hence, C's radius is at least $R(Cl_j \cup \{s\})$ and is strictly less than Z. It follows that $R(Cl_j \cup \{s\}) < Z$, and so C_j is not $Z-$maximal.

□

An Illustrative Example

The solid circle in Figure 4.5 demonstrates circle C_j. The largest, hashed circle represents circle C_j^+. The dot in C_j defines its centre. The line represents the bisector going through the point i encompassed by circle C_j. The location of the external point s' can be seen on the circumference of circle C_j^+.

It is easy to see that $R(Cl_j \cup \{s'\}) = \frac{r_j+Z}{2}$.

If

$$\frac{Z + r_j}{2} < Z,$$

$$\implies Z + r_j < 2Z;$$

As $r_j < Z$, we deduce that $R(Cl_j \cup \{s'\}) < Z$. As this is the largest MCC that can be made with circle C_j united with an external point that is encompassed by C_j^+, it is clear that all other MCCs will have radii $< Z$. Thus a **minimum threshold** of Z has been established. In other words, if there is at least one demand point not

covered by circle C_j which lies within this distance, then the circle cannot be classified as $Z-$maximal as they cannot satisfy condition (2) of Definition 4.2.3.

Let the closure of circle C_j^+ be defined as Cl_j^+. It is worth noting that if $\nexists\ s' \in Cl_j^+$, this does not mean that circle C_j can be defined as a $Z-$maximal circle.

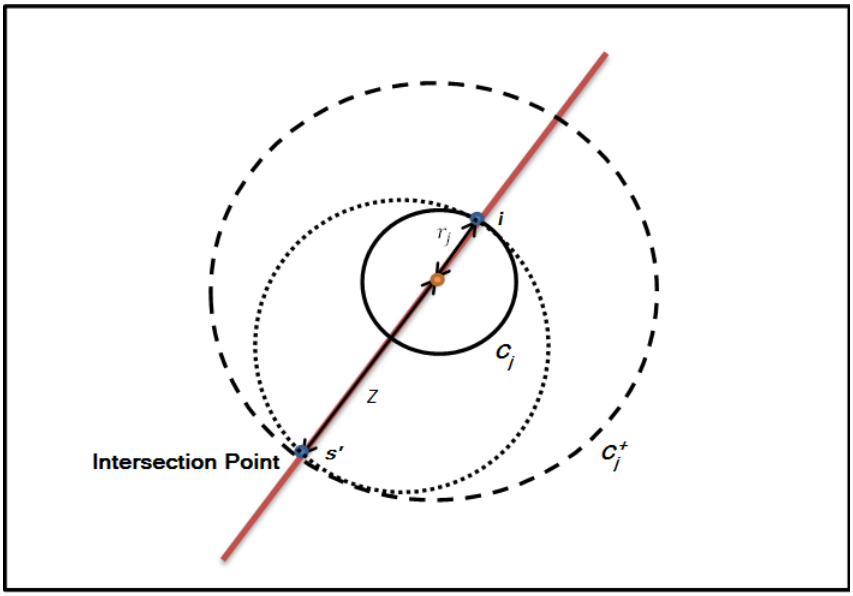


Figure 4.5: Illustrative Example for Enh3

This idea can be further developed by adding a **maximum threshold** of $2Z$ which is stated in Lemma 3.

Lemma 3. *Take any demand point $s \in I$ not covered by C_j. In case $d_{s,j} \geq 2Z$, then $R(Cl_j \cup \{s\}) > Z$.*

Proof. Take $s \in I$ with $d_{s,j} \geq 2Z$. Consider the circle C with centre s and radius $2Z$. As $r_j < Z$, the centre of C_j is not encompassed by C. Therefore, the circle arc of C_j lying within C is strictly less than half C_j's circumference.
But the critical points of C_j span at least half the circle, and so cannot all lie within C. Therefore, $\exists\ i \in Cl_j$ where $d_{i,s} > 2Z$, which implies that $R(Cl_j \cup \{s\}) > Z$. □

Thus if a point that lies at a distance $\geq 2Z$ from (x_{j_c}, y_{j_c}) is added to the set of points encompassed by the circle C_j, the MCC that covers all these points would have a radius $\geq Z$. If this information is known, any point in this area does not need to be checked again and hence computational time can be saved without affecting the quality of the solution.

This enhancement will be referred to as Enh3 and will be incorporated into Step 2 and Step 3 of the FMC algorithm.

Summary

Two observations can be made.

1. If
$$\exists\ i \notin Cl_j \ \mid\ d_{i,j} < Z, \tag{4.12}$$
we can conclude that circle C_j is not $Z-$maximal.

2. If
$$d_{i,j} \geq 2Z \ \ \forall\ i \notin Cl_j \tag{4.13}$$
we can conclude circle C_j is $Z-$maximal.

These two observations led to constructing a *checking area* for circle C_j, say $Check_j$, where
$$Check_j = \{i \notin Cl_j \ \mid\ Z \leq d_{i,j} < 2Z\}. \tag{4.14}$$

If the two observations above are not true and $Check_j \neq \emptyset$, further calculations must be done.

The checking area for circle C_j is displayed in the shaded section of Figure 4.6.

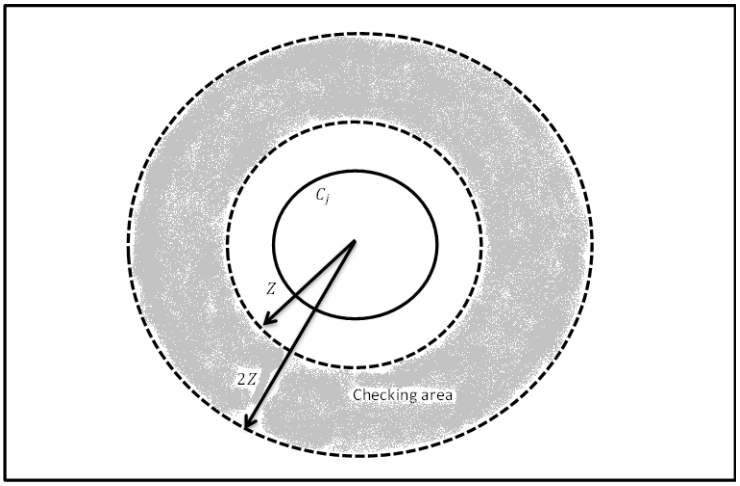


Figure 4.6: Checking Area for circle C_j

An Approximation

If we consider a uniformly distributed set of demand points, we can approximate the number of demand points n' that lies in the checking area. Take A_0 to be the total area and A_c to be the checking area.

We see

$$A_0 = \pi(2Z)^2 = 4\pi Z^2,$$

and

$$A_c = A_0 - \pi Z^2 = 4\pi Z^2 - \pi Z^2 = 3\pi Z^2.$$

Therefore, as

$$\frac{A_0}{A_c} \simeq \frac{n}{n'}.$$

we see

$$\implies n' \simeq n\frac{A_c}{A_0} = \frac{3}{4}n.$$

4.4.4 Enhancement Four: Identifying the Non$-Z-$maximal Circles

If circle C_j is not $Z-$maximal, then there must be a demand point $i \notin Cl_j$ such that $R(Cl_j \cup \{i\}) < Z$. If this point is recorded, in the next iteration this demand point can be the *first to be checked* and hence repeated computations can be discarded. If the MCC of the next iteration is still $< Z$, then we can deduce that this circle is still not $Z-$maximal thus saving computational time. If the MCC is $\geq Z$, we either continue with calculations and conclude it is now classified as $Z-$maximal, or we record the next demand point to cause C_j to be non$-Z-$maximal if it exists. In other words, either way will provide us with useful information that can be used in subsequent iterations.

As an example, say it takes q_j points to find a demand point to determine circle C_j as not $Z-$maximal at iteration t. This means the next iteration starts with the q_j^{th} point instead of starting at the beginning, thus saving the computational time it takes to check the previous $(q_j - 1)$ points.

Let $Start$ be an integer vector of dimension m. The entry $Start_j$ denotes which demand point i should be checked first in the next iteration to see if circle C_j is $Z-$maximal or not. This enhancement, which will be referred to as Enh4, is incorporated into Step 2 and Step 3 of the FMC algorithm.

Summary

If $\exists\ i \in I : R(Cl_j \cup \{i\}) < Z$ at iteration t, then demand point i will be checked first against circle C_j at iteration $t+1$.

4.5 Analysing the $Z-$Maximal Circles-Based Enhancements

The enhancements were first analysed separately so that each one's improvement in computational time could be assessed and its impact measured. For illustrative purposes, the computational times for the separate enhancements for the data set $pr439$ where $p = 70, 80, 90$ and 100 are first shown in Figure 4.7. This is then followed by combining all the refinements together using a certain order that will be based on enhancement performance.

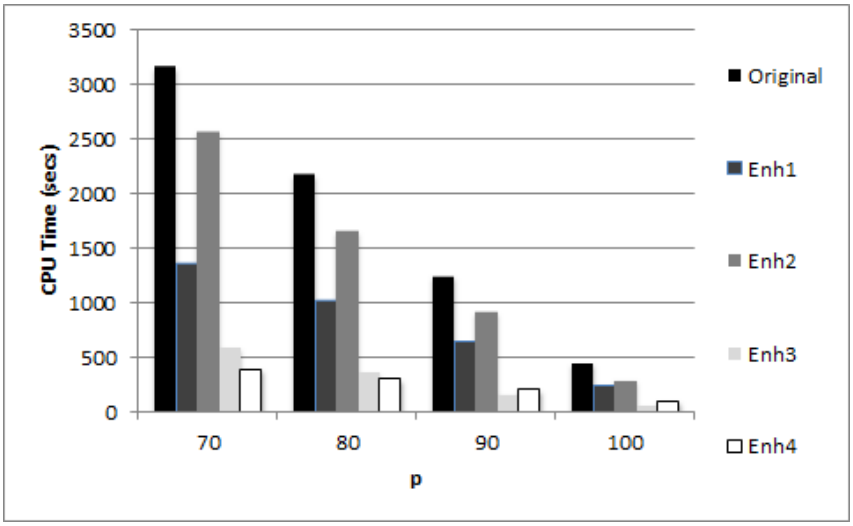


Figure 4.7: Enhancement's Individual Improvements

Individual Performances

Figure 4.7 suggests that the best enhancement, giving an average decrease in computational time of 84.42%, is Enh3. By providing a minimum and maximum threshold

by which the demand points are checked reduces many calculations as many points sit outside of the checking area. Enh4 yields the second best result with an average decrease of 83.26% in computational time. Again, this result is to be expected as Enh4 reduces the number of unnecessary calculations to find all $Z-$maximal circles. By starting at the last known non$-Z-$maximal circle, all previous demand points can be discarded which avoids the unnecessary calculations that they incur. Enh1 is the third best at improving the overall computational time, with an average decrease of 50.65%. This is because this enhancement also reduces the number of calculations by terminating the EHA algorithm earlier whenever possible. Also, by choosing the current critical points as the initial points, the EHA will have less iterations to find the MCC. Finally, Enh2 improves the computational time the least. This is due to not dealing with the $Z-$maximal circle calculations directly; it simply minimises how many circles are needed for these calculations. The average improvement of computational time for Enh2 is 26.26%, which is still significant.

Combined Performance

The four enhancements are embedded into Drezner's original algorithm (using For_{pc}) in our order of individual performances observed earlier which is as follows: Enh3$-$Enh4$-$Enh1 $-$Enh2. To assess the incremental gain of these enhancements, we also conduct the following experiment: in the first run we use Enh3, in the second we use Enh3 and Enh4, and in the third Enh3, Enh4 and Enh1 are used. The fourth run consists of the proposed algorithm with all the enhancements incorporated. The results are shown in Figure 4.8.

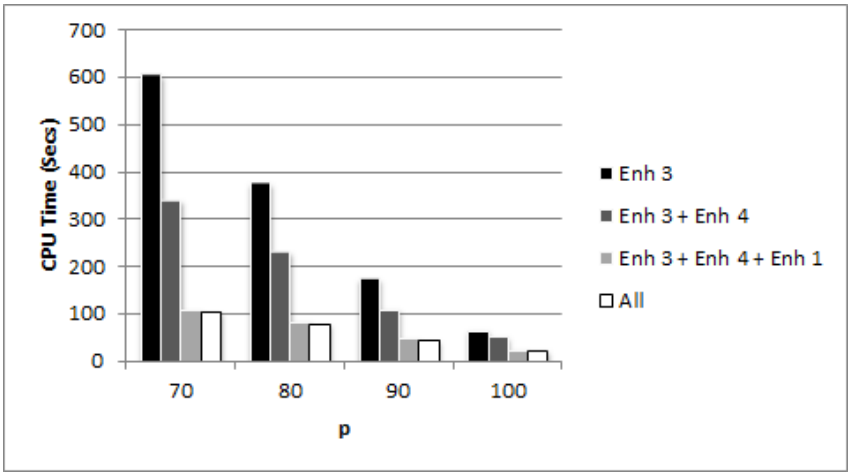


Figure 4.8: Comparison on CPU Time for the Enhancements

It is clear to see that the enhancements greatly improve the computational time. The first enhancement reduces the total computational time by an average of 84.49% as previously mentioned, and by adding Enh4 this is decreased further to 90.26%. After the addition of Enh1, the average decrease becomes 96.46% and finally with all enhancements this reaches 96.71%. In other words, just above 3% of the original computational time is needed on average, leading to a very strong result.

It is also worth noting that the incremental decrease in computational time is not directly additive as there is a high level of association between their individual contributions. For instance, after gaining 84% with Enh3, one might expect Enh4 to yield 83% of the remaining 16%. This would therefore give a new decrease of approximately 97%. However, it only decreases it to just over 90% (i.e., an extra 5.8% only).

Section 4.6 describes the enhanced reverse relaxation with all four enhancements added to the algorithm.

4.6 The Complete Revised Optimal Algorithm

The full revised FMC algorithm is given in Figure 4.9. It is similar to the original FMC algorithm, except Steps 2 and 3 in Figure 4.2 have been modified accordingly to accommodate the enhancements described in this study.

The Drezner enhanced algorithm (DEA) is similar to the Drezner's original algorithm stated previously in Figure 4.1, except that in Step 5 the formulation For_{pc} is used instead of $For_0^{(a)}$ or $For_0^{(b)}$ and an extra step (Step 3 shown in Figure 4.10) has been added to accommodate the enhancements. For completeness, we reproduce the full DEA in Figure 4.10.

Section 4.7 displays the new results for $pr439$ and $rat575$ with all four enhancements incorporated.

Step 1: Input vectors $Start_j$ and $CircMax_j$. Set $J_Z = \emptyset$.

Step 2: For all $C_j \in J$, if $CircMax_j = 1$, add C_j to J_Z (i.e. set $J_Z = J_Z \cup C_j$) and remove C_j from the set $\{C_J^1 \cup C_J^2 \cup C_J^3\}$.

Step 3: (Find all $Z-$maximal circles in the set C_J^1.)
For all $C_j \in C_J^1$ where $CircMax_j = 0$ do:
Take the circle C_j. Starting from demand point $Start_j$, take $l \in I$ and compute $d_{l,j}$. If $\exists\ l \in I$ such that $\frac{d_{l,j}}{2} < Z$, the circle is not $Z-$maximal. Discard C_j from further investigation and set $Start_j = l$.
Else, the circle is $Z-$maximal by definition. Add C_j to J_Z (i.e. $J_Z = J_Z \cup C_j$) and set $CircMax_j = 1$.

Step 4: (Find all $Z-$maximal circles in the set $C_J^2 \cup C_J^3$.)
For all $C_j \in \{C_J^2 \cup C_J^3\}$ where $CircMax_j = 0$ do:

Step $4A$:

(i) Starting from demand point $Start_j$, if $\exists\ i \notin Cl_j$ where $d_{i,j} < Z$, go to Step $4B(ii)$, else go to $4A(ii)$.

(ii) Starting from demand point $Start_j$, if $\forall\ \ i \notin Cl_j,\ d_{i,j} \geq 2Z$, go to Step $4B(i)$, else go to $4A(iii)$.

(iii) While $\exists\ i \notin Cl_j$ with $Z \leq d_{i,j} < 2Z$, starting from demand point $Start_j$ do:
Use the EHA to find $R(Cl_j \cup \{i\})$.
If a circle with radius $\geq Z$ is found at any point during the EHA, go back to start of $4A(iii)$ starting from the next i value, else continue to find $R(Cl_j \cup \{i\})$. If $R(Cl_j \cup \{i\}) < Z$, then C_j is not $Z-$maximal: go to Step $4B(ii)$.

(iv) If this point is reached the circle is maximal: go to Step $4B(i)$.

Step $4B$:

(i) (Circle C_j is $Z-$maximal by definition.) Add C_j to J_Z (i.e. set $J_Z = J_Z \cup C_j$) and set $CircMax_j = 1$.

(ii) (Circle C_j is not $Z-$maximal by definition.) Set $Start_j = i$.

Figure 4.9: The FMC-Revised Algorithm

4.7 Computational Results

Tables 4.5 & 4.6 show the results found for the data sets $pr439$ and $rat575$ using the DEA. The columns are organised in the same way as in the previous tables. As a summary, we also produced Table 4.7 to show the new and the old duration for each value of p including the percentage decrease. Note that these computational times do

Step 1. Find the circles made from one, two or three demand points. This creates three sets of circles : C_J^1, C_J^2 and C_J^3. Discard any circle in C_J^3 whose three points create an obtuse or right-angled triangle.

Step 2. Find an initial solution and set the solution value as the initial upper bound, Z.

Step 3. Set $Start_j = 1$ and $CircMax_j = 0$ for $j = 1, \ldots, m$.

Step 4. Eliminate all circles whose radii are $\geq Z$ from C_J^2 and C_J^3.

Step 5. Find all $Z-$maximal circles using the FMC-Revised algorithm with the threshold Z (Figure 4.9). Let J_Z be the set of $Z-$maximal circles.

Step 6. Solve For_{pc} using the set of current $Z-$maximal circles J_Z.
If a solution is found, set Z to be the new upper bound, $J_Z = \emptyset$ and go back to Step 4.
Else, set the upper bound Z is the optimal solution value of the continuous $p-$centre problem and stop.

Figure 4.10: Drezner Enhanced Algorithm (DEA)

not include the computational time for the H_2 heuristic.

It is clear to see the enhanced method has greatly reduced the computational time for both data sets. Whereas it took the data set $pr439$ just over 4 hours average computational time before, it now takes an average time of just over 12 minutes leading to a massive reduction of 96%.

For the data set $rat575$, the computational time has also been reduced. For the smaller values of p (10, 20 and 30), the majority of the time was taken computing the $Z-$maximal circles leading to a reduction of over 90%. However, for the other values of p, the majority of the computational time is taken up solving the problem in CPLEX, leading to a relatively small though still significant overall reduction of nearly 50%. The next section will investigate a new policy developed to reduce the time spent solving For_{pc}.

	H_2 Heuristic		Optimal Solution						
p	Z_1	CPU Time	Z^*	Loop CPU Time (secs) [a]	# Loops	Maxi Circles (secs)	CPLEX (secs)	Maxi Circles (%)	CPLEX (%)
10	1716.510	96.88	1716.510	342.78	2	278.96	34.52	81.38	10.07
20	1169.540	170.28	1029.710	2856.38	36	359.05	282.05	12.57	9.87
30	975.000	205.36	739.193	2146.67	49	229.60	207.87	10.70	9.68
40	874.271	218.9	580.005	1515.29	67	171.14	200.49	11.29	13.23
50	580.005	235.61	468.542	159.49	38	21.90	51.09	13.73	32.04
60	570.088	246.86	400.195	170.38	48	23.24	53.20	13.64	31.22
70	503.271	256.30	357.946	97.63	47	13.77	36.71	14.11	37.60
80	467.039	300.01	312.500	73.52	52	9.61	31.62	13.07	43.02
90	391.511	276.20	280.903	38.01	48	4.71	20.85	12.39	54.86
100	315.486	332.53	256.680	16.77	32	1.50	11.06	8.93	65.93
Average	**756.272**	**233.90**	**614.218**	**741.69**	**42**	**111.35**	**92.95**	**19.18**	**30.75**

[a] This excludes computational time for the H_2 heuristic.

Table 4.5: $n = 439$ TSP-Lib with Enhancements

	H_2 Heuristic		Optimal Solution						
p	Z_1	CPU Time	Z^*	Loop CPU Time (secs) [a]	# Loops	Maxi Circles (secs)	CPLEX (secs)	Maxi Circles (%)	CPLEX (%)
10	69.426	98.34	67.926	5572.02	10	693.86	336.28	12.45	6.04
20	48.107	175.62	45.475	1616.05	11	109.75	495.80	6.79	30.68
30	39.655	238.26	35.556	1023.14	14	46.20	544.21	4.51	53.19
40	33.365	296.90	30.063	37660.80	11	17.41	37514.80	0.05	99.61
50	30.336	403.76	25.826	6352.86	15	12.85	6247.59	0.20	98.34
60	27.951	422.18	23.163	26870.00	18	9.26	26800.50	0.03	99.74
70	25.578	558.85	20.858	26123.80	19	6.22	26082.30	0.02	99.84
80	24.135	535.90	19.026	32343.20	17	4.41	32343.20	0.01	99.91
90	21.932	743.20	17.460	2167.610	18	3.04	2149.99	0.14	99.19
100	20.402	795.13	16.420	25074.40	15	1.93	25074.40	0.01	99.95
Average	**34.089**	**426.81**	**30.177**	**16480.39**	**15**	**90.49**	**15758.90**	**2.42**	**78.65**

[a] This excludes computational time for the H_2 heuristic.

Table 4.6: $n = 575$ TSP-Lib with Enhancements

	$pr439$			$rat575$		
p	Original Loop CPU Time (secs)[a]	New Loop CPU Time (secs)[a]	Percentage Decrease (%)	Original Loop CPU Time (secs)[a]	New Loop CPU Time (secs)[a]	Percentage Decrease (%)
10	6252.72	342.78	94.52	83898.60	5572.02	93.36
20	56753.00	2856.38	94.97	19087.6	1616.05	91.53
30	37017.10	2146.67	94.20	9743.91	1023.14	89.50
40	31355.00	1515.29	95.17	41733.00	37660.80	9.76
50	4939.25	159.49	96.77	9612.60	6352.86	33.91
60	4956.45	170.38	96.56	28344.00	26870.00	5.20
70	3170.89	97.63	96.92	40256.90	26123.80	35.11
80	2186.27	73.52	96.64	40181.70	32343.20	19.51
90	1258.22	38.01	96.98	4260.10	2167.61	49.12
100	462.30	16.77	96.37	33694.00	25074.40	25.58
Average	**14835.12**	**741.69**	**95.91**	**31081.24**	**16480.39**	**45.26**

[a] This excludes computational time for the H_2 heuristic.

Table 4.7: Original vs. Revised Drezner's algorithm for $n = 439$ TSP-Lib and $n = 575$ TSP-Lib

4.8 A Compromise Solution in CPLEX

In this section, we investigate how to balance the time spent between computing the $Z-$maximal circles and the level of the solution quality which is considered acceptable when solving For_{pc}. However, to guarantee optimality we need to show at one stage that For_{pc} has no feasible solution, and hence the final iteration needs to run to the very end. In other words, it is not possible to reduce computational time here by terminating the search earlier in the last run.

As a demonstration, Table 4.8 shows the total time taken in CPLEX compared to the time consumed in the last iteration in CPLEX (i.e. the amount of time required for CPLEX to find that no feasible solution exists). Figure 4.11 displays the average computational time spent during each iteration compared to the amount of time consumed by CPLEX in the last iteration. Though a relatively considerable amount of time is used in the last iteration (accounting for approximately 10-20% of the total computational time), the computational time taken in the previous iterations is nonetheless worth exploring for possible improvement. A compromise feasible solution to save computational time in CPLEX while limiting the total number of iterations of the entire algorithm will be the focus of in this section.

p	CPLEX Loop CPU Time (secs)	CPLEX Final Iteration CPU Time (secs)	Loops	Percentage Use (%)	Average(%) per Loop w/o last iteration	Average (%) per Loop
10	336.28	31.81	10	9.46	11.11	10.00
20	495.80	105.96	11	21.37	10.00	9.09
30	544.21	68.25	14	12.54	7.69	7.14
40	37514.80	12789.40	11	35.81	10.00	9.09
50	6247.59	673.42	15	10.78	7.15	6.67
60	26800.50	3821.60	18	14.26	5.88	5.56
70	26082.30	2231.55	19	8.56	5.56	5.26
80	32343.20	647.03	17	2.00	6.25	5.88
90	2149.99	41.48	18	1.93	5.88	5.56
100	25074.40	4577.57	15	18.26	7.14	6.67
Average	**15758.91**	**2498.81**	**15**	**15.86**	**7.14**	**6.67**

Table 4.8: CPLEX Durations (secs) for both the total and the last iteration in the case of $n = 575$ TSP-Lib

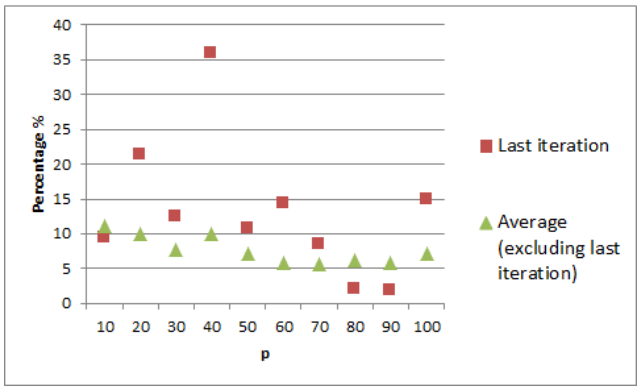


Figure 4.11: Average computational time % in CPLEX per iteration vs. last iteration for $rat575$

There are several ways in which the search can be terminated early in previous runs whilst producing a feasible solution for For_{pc}. An example would be to impose a time limit, however one cannot guarantee that a feasible solution will be found within the time and so other options are investigated.

One option would be to use either a) the set covering formulation $For_0^{(a)}$ or b) the $p-$centre formulation where the first feasible solution of For_{pc} is used (i.e. equivalent to $For_0^{(b)}$) with our enhanced algorithm, as both of these were initially suggested by Drezner. These two possibilities were initially discarded as they greatly increased both the number of iterations and the total computational time due to the time spent calculating the $Z-$maximal circles. However, as the computational time for the revised

method has now dramatically improved, these options may have become viable to warrant further examination.

The set covering formulation was tested with the Drezner enhanced algorithm for data sets $pr439$ and $rat575$ where $p = 100$ and was still found to be inefficient. Results for these two instances can be found in Table A.1 in Appendix A.

Drezner's alternative suggestion of updating the upper bound as the first feasible solution found in each iteration was tested with the Drezner enhanced algorithm for data sets $pr439$ and $rat575$, and the results can be found in Table A.2 and A.3 in Appendix A. Interestingly, this method was found to have some strengths as well as weaknesses. Among the strengths is that the computational times were improved for $n = 575$, including the computational time spent in CPLEX. This is significant as the total computational time for $rat575$ was largely affected by the time spent in CPLEX. However, the number of iterations for the algorithm was still extremely large which we aim to avoid here by adopting a different strategy.

Our strategy manipulates the duality gap so that CPLEX terminates earlier with a good feasible, but not necessarily optimal, solution whenever it manages to find at least one. However, the value of the duality gap can be both sensitive and critical which can make our algorithm less robust. The algorithm cannot terminate too early as it could simply increase the number of iterations greatly, and therefore increase the time spent computing the $Z-$maximal circles. It is therefore important to devise a reasonable compromise. In this study, we propose the following self−learning CPLEX policy which takes into consideration information from previous iterations.

It is worth noting that the following duality gap policy is only implemented when CPLEX finds at least one feasible solution in any run of CPLEX. If no feasible solution has been identified in a given run, CPLEX continues until the maximum time limit is reached where the search terminates. Hence the obtained Z value of the previous run is used as the final solution (which obviously cannot be guaranteed to be optimal).

After the first results, it was noted that terminating the program before 1.5% would be

too early. It may reach this duality gap value quickly, but the feasible solution found often deviated far from the optimal solution. This would greatly increase the number of overall iterations, thus causing a similar weakness to those mentioned using the first feasible solution.

A maximum duality gap value of 1% should be appropriate in most cases. However, for some values of p (e.g. $p = 10$) the data spends far more time computing the $Z-$maximal circles rather than solving the problem. If a duality gap was increased from 0% to 1% in this instance, Z would be updated more frequently which would lead to the undesirable outcome of calculating the $Z-$maximal circles more often. To overcome this, a self-learning policy is constructed where the selection of the duality gap value is based on previous iterations.

4.8.1 An Adaptive CPLEX Policy

At iteration t, the moving average for the computational time for calculating $Z-$maximal circles (T_{Max}) and solving the problem in CPLEX (T_{CPLEX}) based on the last μ iterations is respectively defined as follows.

$$G_t^{\mu}(A) = \frac{\sum_{t'=t-\mu}^{t} A^{t'}}{\mu} \tag{4.15}$$

where $A = \{T_{Max}, T_{CPLEX}\}$, and $A^{t'}$ is the corresponding time at iteration t'.
We define μ as

$$\mu = \begin{cases} \frac{t}{2} \text{ if } t \geq K, \\ t \text{ else.} \end{cases}$$

In other words, the classical average is used if $t < K$, otherwise the moving average over half of the past iterations is adopted. In this study, we used $K = 6$ based on preliminary results.

We use the following scheme based on the performance ratio $\xi = \frac{G_t^{\mu}(T_{Max})}{G_t^{\mu}(T_{CPLEX})}$;

a) If

$$\xi \geq 1 \tag{4.16}$$

then the time for computing the $Z-$maximal circles is much larger than the time spent solving the problem in CPLEX. Therefore, the number of iterations need to be reduced as much as possible, and so we set the duality gap to 0%.

b) However, if

$$\xi \leq 0.4 \tag{4.17}$$

then the majority of the computational time is spent solving the problem in CPLEX, and therefore we wish to exit CPLEX sooner with a feasible solution rather than seeking an optimal one. The duality gap is set to be 1%.

c) If ξ has any other value, then the computational times are considered to be more or less similar. In this case, we wish to reach a balance between finding the optimal solution and leaving CPLEX early, hence we set the duality gap to be 0.5%.

In summary, the following conditions related to the duality gap are given.

$$\text{Duality Gap}=\begin{cases} 0 & \text{if } \xi \geq 1, \\ 0.5\% & \text{if } 0.4 < \xi < 1, \\ 1\% & \text{if } \xi \leq 0.4. \end{cases} \tag{4.18}$$

This policy, which uses adaptive learning, is less sensitive to the effect of the data's distribution on the computational time as therefore it is very reliable.

It is important to explain why this policy (4.18) chooses the moving average to analyse the past computational time is chosen. The total average was considered, but this was found unreliable as it includes results from very early iterations that could behave differently to the current iterations. Successive iterations can behave similarly, and so this suggests the previous iterations would give the best indication as to which duality gap value to use. However, this method has a weakness. If a previous time was randomly very extreme, as this will affect the overall computational time. Thus, the moving average was chosen as it finds a balance between accounting for extreme values whilst only considering recent computational times. Furthermore, by choosing $t' = \frac{t}{2}$, this means the range of the moving average increases as the iterations increase, whilst

keeping the values in the range as the most recent.

Only three values were assigned as cut-off points to establish which duality gap to choose, and it is noted that many more values could be considered and analysed for further work. Additionally, the largest duality gap that can be allocated is 1%. However other larger duality gaps, such as 2.5% and 5%, could also be analysed.

4.8.2 Results with the Adaptive CPLEX Policy

The results for $rat575$, that include the CPLEX adaptive policy, are found in Table 4.9, displayed alongside the total computational time required to optimally solve this data set using the enhanced algorithm without the duality gap policy. Results show that the average decrease in computational time is now 72.91% from the original CPU times, and it has decreased a further 50.05% from this new computational time when incorporating the adaptive CPLEX policy with the enhancements. This is a promising result and demonstrates that the adaptive CPLEX policy has a large and positive effect on the overall duration.

Although it already had promising results, the data set $pr439$ was also tested with the policy. This can be found in Table A.4 in Appendix A. The results are still extremely promising, however the overall computational time is slightly worse than previously. This is because the total time spent in CPLEX is still larger compared to the time spent computing the $Z-$maximal circles. This therefore sets the duality gap to be 0.5% or 1%, thus increasing the number of iterations and the overall computational time. Furthermore, this is also the reason as to why there is a slight increase in computational time for the data set $rat575$ where $p = 10 - 30$.

However, the purpose of the adaptive CPLEX policy is to find a compromise for all data sets so that it can be applied to any distribution type. As the overall computational time is still decreased by 94.5% for $pr439$ compared to the original results, it is still considered an extremely encouraging result.

Furthermore, it is important to recognise that for some values of p (e.g. $p = 10$) the total duration could be slightly increased if the majority of time is spent comput-

	H_2 Heuristic	Optimal Solution									
p	Z_1	Z^*	Loop CPU Time w/o Duality Gap (secs)[a]	Loop CPU Time (secs) [a]	Percentage Decrease (%)	# Loops	Max Circles (secs)	CPLEX (secs)		Max Circles (%)	CPLEX (%)
								Total	Last Loop		
10	69.426	67.926	5572.02	5732.12	-2.86	10	690.69	340.37	32.89	12.05	5.93
20	48.107	45.475	1616.05	1634.74	-1.15	11	112.83	471.74	108.68	6.90	28.86
30	39.655	35.556	1023.14	1254.57	-22.62	30	58.88	730.55	69.97	4.69	58.23
40	33.365	30.063	37660.80	25949.90	31.10	15	19.55	25793.20	12936.20	0.08	99.40
50	30.336	25.826	6352.86	3161.59	50.23	23	14.17	3052.89	675.08	0.45	96.56
60	27.951	23.163	26870.00	9134.14	66.01	29	10.49	9063.42	3733.39	0.11	99.26
70	25.578	20.858	26123.80	15961.50	38.91	24	6.53	15920.30	2219.57	0.04	99.74
80	24.135	19.026	32372.30	5656.99	82.53	74	8.80	5619.18	642.85	0.16	99.33
90	21.932	17.460	2167.61	996.43	54.03	34	3.98	976.77	41.86	0.40	98.03
100	20.402	16.420	25086.30	12862.90	48.73	23	2.29	12850.30	4614.62	0.02	99.90
Average	**34.089**	**30.178**	**16484.48**	**8234.49**	**50.05**	**27**	**92.82**	**7481.87**	**2507.51**	**2.49**	**78.52**

[a] This excludes computational time for the H_2 heuristic.

Table 4.9: $n = 575$ TSP-Lib with Enhancements and Adaptive CPLEX Policy

ing the $Z-$maximal circles. This is because in the first iteration, there is no way of knowing whether the majority of time will be spent on computing $Z-$maximal circles or solving the problem in CPLEX due to the fact that the problem is yet to be solved in CPLEX for the first time. To respond to this issue, the duality gap is set to 0.5% in the first iteration as a compromise, which in this instance could cause an extra iteration or more. If this is the case, and the majority of time is spent computing the $Z-$maximal circles, the total duration will be a little higher.

4.8.3 The Adaptive CPLEX Policy where $\mu = 1$

A simpler test was attempted where $\mu = 1$ (i.e. the duality gap was determined by the results from the previous iteration). The results for $pr439$ and $rat575$ using this simpler adaptive CPLEX policy can be found in Tables 4.10 and 4.11.

	H_2 Heuristic	Optimal Solution							
p	Z_1	Z^*	Loop CPU Time (secs) [a]	# Loops	Max Circles (secs)	CPLEX (secs)		Max Circles (%)	CPLEX (%)
						Total	Last Loop		
10	1716.510	1716.510	359.00	2	293.31	35.86	17.81	81.70	9.99
20	1169.540	1029.710	2927.82	36	371.49	288.74	7.07	12.69	9.86
30	975.00	739.193	2165.59	49	232.89	210.5	3.23	10.75	9.72
40	874.271	580.005	1598.51	88	184.49	247.93	1.84	11.54	15.51
50	580.005	468.542	360.06	177	47.35	207.65	2.10	13.15	57.67
60	570.088	400.195	322.27	182	42.23	171.44	0.71	13.10	53.20
70	503.271	357.946	193.09	162	24.76	112.18	0.55	12.82	58.09
80	467.039	312.500	141.15	165	16.62	86.19	0.36	11.78	61.06
90	391.511	280.903	94.86	165	10.02	67.54	0.32	10.56	71.19
100	315.486	256.680	29.39	69	2.63	21.41	0.38	8.94	72.85
Average	**756.272**	**233.900**	**819.18**	**110**	**122.58**	**144.95**	**3.44**	**18.70**	**41.92**

[a] This excludes computational time for the H_2 heuristic.

Table 4.10: Results for $n = 439$ TSP-Lib with Enhancements and the Adaptive CPLEX Policy where $\mu = 1$

Results show that this simpler approach led to an inferior outcome. For example, when studying the data set $rat575$, it can be seen that more computational time was spent solving the problem in CPLEX compared to the results found in Table 4.9. As we want to decrease the time spent in CPLEX, this was an undesired affect. Therefore, it was decided that the more informative method of past iterations (i.e. using the moving average) should be kept.

	H_2 Heuristic	Optimal Solution							
p	Z_1	Z^*	Loop CPU Time (secs) [a]	# Loops	Max Circles (secs)	CPLEX (secs)		Max Circles (%)	CPLEX (%)
						Total	Last Loop		
10	69.426	67.926	5631.73	10	697.62	333.15	31.96	12.39	5.92
20	48.107	45.475	1550.06	11	110.59	427.93	106.01	7.13	27.61
30	39.655	35.556	1288.47	37	62.03	778.10	69.33	4.81	60.39
40	33.365	30.063	21914.60	13	18.39	21765.90	13057.90	0.08	99.32
50	30.336	25.826	3213.43	23	14.33	3105.54	697.21	0.45	96.64
60	27.951	23.163	8517.14	30	10.64	8446.75	3831.13	0.12	99.17
70	25.578	20.858	23648.60	31	7.23	23606.10	2411.24	0.03	99.82
80	24.135	19.026	6215.33	44	6.84	6180.39	705.50	0.11	99.44
90	21.932	17.460	1075.65	34	4.05	1055.75	45.21	0.38	98.15
100	20.402	16.420	14149.70	23	2.39	14136.50	5167.62	0.02	99.91
Average	**34.089**	**426.810**	**8720.47**	**26**	**93.41**	**7983.62**	**2612.31**	**2.55**	**78.64**

[a] This excludes computational time for the H_2 heuristic.

Table 4.11: Results for $n = 575$ TSP-Lib with Enhancements and the Adaptive CPLEX Policy where $\mu = 1$

4.9 Overall Computational Results

The performance of the DEA is now fully evaluated under two scenarios. It is important to note that from this point, the DEA will refer to the Drezner enhanced algorithm paired with the adaptive CPLEX policy. In Scenario One, the large known TSP-Library data sets $rat575$, $rat783$, $pr1002$ and $rl1323$ are tested, and in Scenario Two, 3 new types of large data sets, namely randomly spread, semi-clustered and clustered, are generated and tested.

4.9.1 Scenario One: Results using the TSP-Library data sets

As these are very large data sets, a total time limit of 24 hours was given to each value of p. If the algorithm at some value p takes longer than 24 hours to run, the program is terminated and the upper bound at that time is recorded as the best solution found.

Tables 4.12 − 4.15 are arranged similarly to the tables in Section 4.3, except the initial upper bound value, Z_H, is now the best known heuristic value (see Elshaikh *et al* (2016)) as we aim to obtain the optimal solution. Therefore, in this experiment, we deviate from the method previously used where an initial upper bound was obtained

using the H_2 heuristic. Note that the computational times given here do not include this heuristic step, but the times can be collected from Elshaikh *et al* (2016).

	Best Heuristic	Optimal Solution							
p	Z_H	Z^*	Loop CPU Time (secs) [a]	# Loops	Max Circles (secs)	CPLEX (secs)		Max Circles (%)	CPLEX (%)
						Total	Last Loop		
10	67.926	67.926	489.53	1	413.20	32.37	32.37	84.41	6.612
20	45.621	45.475	384.79	3	49.50	272.52	107.70	12.86	70.82
30	35.556	35.556	87.16	1	11.19	68.84	68.84	12.83	78.99
40	30.265	30.063	20898.30	5	6.57	20880.01	13085.80	0.03	99.91
50	26.173	25.826	2476.32	10	4.35	2462.60	670.71	0.18	99.45
60	23.622	23.163	8888.40	12	3.03	8878.01	3749.88	0.03	99.88
70	21.059	20.858	16283.70	9	1.64	16277.80	2238.12	0.01	99.9
80	19.510	19.026	3893.66	13	1.45	3887.75	646.53	0.04	99.85
90	17.923	17.460	868.39	18	1.22	863.18	41.75	0.14	99.40
100	16.551	16.420	13268.80	8	0.55	13265.40	4626.44	0.00	99.97
Average	**30.421**	**30.178**	**6753.90**	**8**	**49.27**	**6688.86**	**2526.81**	**11.05**	**85.49**

[a] This excludes computational time for the heuristic step.

Table 4.12: Results for $n = 575$ TSP-Lib using Enhancements and CPLEX Adaptive Policy starting from Best Heuristic Value

	Best Heuristic	Optimal/Best Solution							
p	Z_H	Z^*	Loop CPU Time (secs) [a]	# Loops	Max Circles (secs)	CPLEX (secs)		Max Circles (%)	CPLEX (%)
						Total	Last Loop		
10	79.313	79.313	5696.39	2	2918.48	978.14	402.57	51.23	17.17
20	53.441	53.332	2884.05	8	224.16	2410.67	400.08	7.77	83.59
30	42.395	42.307	21833.60	4	55.52	21714.00	13229.40	0.25	99.45
40	35.962	$35.861^{\perp}$	86400.00	1	19.30	86380.00	86370.00	0.02	99.98
50	31.184	$31.041^{\perp}$	86400.00	10	14.81	86355.50	33887.70	0.01	99.95
60	28.053	$27.880^{\perp}$	86400.00	14	10.95	86365.10	80032.39	0.01	99.96
70	25.446	$25.239^{\perp}$	86400.00	3	4.21	86381.60	39254.10	0.004	99.98
80	23.560	$23.192^{\perp}$	86400.00	9	5.43	86384.24	1530.90	0.006	99.98
90	21.710	$21.319^{\perp}$	86400.00	12	5.01	86384.30	54352.70	0.005	99.98
100	20.334	$19.999^{\perp}$	86400.00	7	2.03	86387.10	50190.10	0.002	99.99
Average	**36.140**	– – –	– – –	**7**	**325.99**	**62974.05**	**35964.00**	**5.94**	**90.00**

[a] This excludes computational time for the heuristic step.

$^{\perp}$ Best feasible solution found within 86400 seconds.

Table 4.13: Results for $n = 783$ TSP-Lib using Enhancements and CPLEX Adaptive Policy starting from Best Heuristic Value

Tables 4.12-4.15 provide extra information regarding the computational time spent in

CPLEX. In order to establish how much computational time cannot be improved on (i.e. the computational time consumed in the last iteration) the column representing time spent in CPLEX is now divided into two, with one half showing the total time spent in CPLEX and the other half showing how long the last iteration took in CPLEX. Therefore, in the instance where the algorithm reaches the maximum time limit, the result in the second half of this column may not be showing the time spent to reach optimality. However, in each of these circumstances, no further feasible solution was found in the final iteration (except for the case where $n = 783, p = 40$). Thus, this indicates that the solution found in the previous iteration may be the optimal solution.

Furthermore, in the instance where $n = 783$ and $p = 40$, a feasible solution was found but the adaptive CPLEX policy value had not been reached (i.e. 0%, 0.5% or 1%). The program was therefore allowed to run for a further hour (with the solution found at this iteration as its new upper bound) to see if this solution could be improved. However, during this time no further feasible solution was found, thus showing that the last feasible solution found could be optimal.

	Best Heuristic	Optimal Solution							
p	Z_H	Z^*	Loop CPU Time (secs) [a]	# Loops	Max Circles (secs)	CPLEX (secs)		Max Circles (%)	CPLEX (%)
						Total	Last Loop		
10^+	2389.360	–	–	–	–	–	–	–	–
20	1609.540	1607.530	4904.66	10	825.07	2786.07	340.83	16.82	56.80
30	1231.360	1231.360	881.26	1	86.42	739.83	739.83	9.81	83.95
40	1030.400	1021.410	1778.08	29	121.62	1404.82	190.49	6.84	79.01
50	901.455	895.342	13011.90	12	42.29	12867.60	353.84	0.33	98.89
60	801.474	795.709	8961.03	22	40.29	8843.69	785.27	0.45	98.69
70	727.154	725.431	1502.26	3	10.86	1458.29	1436.05	0.72	97.07
80	664.798	655.746	917.42	15	16.35	853.75	78.91	1.78	93.06
90	604.152	604.152	373.52	1	4.20	349.55	349.55	1.12	93.58
100	559.017	555.662	123.78	10	6.82	91.64	12.70	5.51	74.04
Average	**1051.870**	**825.540**	**3605.99**	**11**	**128.21**	**3266.13**	**476.39**	**4.82**	**86.12**

[a] This excludes computational time for the heuristic step.
[+] Could not be computed due to computer memory issues.

Table 4.14: Results for $n = 1002$ TSP-Lib using Enhancements and CPLEX Adaptive Policy starting from Best Heuristic Value

	Best Heuristic	Optimal/Best Solution							
p	Z_H	Z^*	Loop CPU Time (secs) [a]	# Loops	Max Circles (secs)	CPLEX (secs)		Max Circles (%)	CPLEX (%)
						Total	Last Loop		
10^+	2897.490	–	–	–	–	–	–	–	–
20^+	1886.820	–	–	–	–	–	–	–	–
30	1466.970	1466.970	29522.00	2	1605.09	26403.90	12725.60	5.43	89.44
40	1236.380	1235.660$^\perp$	86400.00	5	199.23	86150.77	19277.17	0.23	99.71
50	1060.820	1060.420$^\perp$	86400.00	2	48.08	85933.90	400.00	0.06	99.46
60	941.870	940.483$^\perp$	86400.00	7	43.10	86333.90	18895.60	0.05	99.90
70	844.967	843.801	13454.40	12	38.72	13323.10	6278.02	0.29	99.02
80	774.764	774.764	51229.30	1	9.45	51164.10	51164.10	0.02	99.87
90	720.625	706.145	5942.07	33	46.91	5750.88	119.51	0.80	96.78
100	662.936	658.997	37388.90	15	20.53	37273.30	6915.90	0.05	99.69
Average	**1249.364**	– – –	– – –	**10**	**251.39**	**49041.73**	**14471.99**	**0.87**	**97.98**

[a] This excludes computational time for the heuristic step.
$^\perp$ Best feasible solution found within 86400 seconds.
$^+$ Could not be computed due to computer memory issues.

Table 4.15: Results for $n = 1323$ TSP-Lib using Enhancements and CPLEX Adaptive Policy starting from Best Heuristic Value

It is important to note that for smaller values of p (i.e. $p = 10$ for $pr1002$ and $p \leq 20$ for $rl1323$) computer memory becomes an issue leading to no results being found. This could be due the initial upper bound being higher in these instances, leading to a relatively large number of circles being considered and thus making the ILP model too big to be handled.

The results show that the DEA can now find very good, and even optimal, solutions for these large TSP-Library data sets. Optimal solutions are found for the first time for the larger data sets $pr1002$ and $rl1323$ where a reasonable amount of computational time is required. This promising result therefore displays the DEA's dominance for solving the continuous $p-$centre problem. The results also suggest that the data's distribution, rather than size, contributes to how easily the problem can be solved. For example, the data set $rat783$, whose distribution is much like $rat585$, was more challenging to solve optimally. However, the data set $pr1002$, whose distribution is similar to $pr439$, was less challenging. This observation will be analysed further in the next section.

4.9.2 Scenario Two: Results using our new generated data sets

This section aims to study how efficiently the DEA solves data sets with distinguished contrasting distributions. Clustered, semi-clustered and randomly distributed data sets were generated using $C++$ programming language where $n = 400, 600$ and 800. Figures 4.12, 4.13 and 4.14 show the clustered, semi-clustered and random distributions respectively for $n = 400, 600$ and 800.

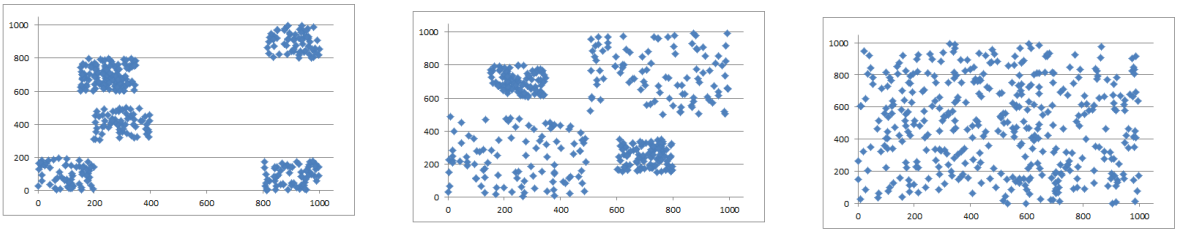


Figure 4.12: Distributions where $n = 400$

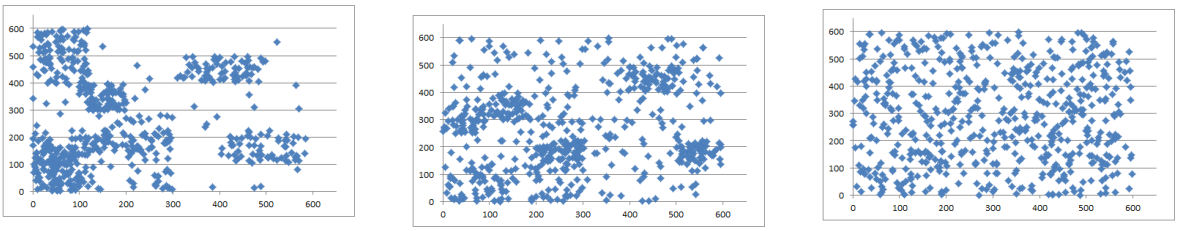


Figure 4.13: Distributions where $n = 600$

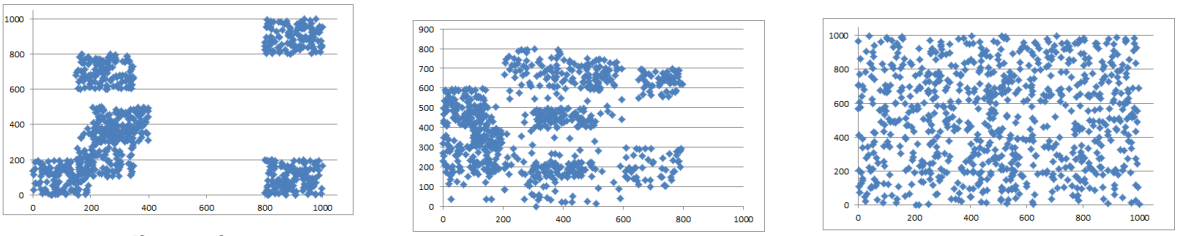


Figure 4.14: Distributions where $n = 800$

Tables 4.16, 4.17 and 4.18 gives the optimal results for each data set for $p = 25, 50, 75$ and 100 where $n = 400, 600$ and 800 respectively. The first column displays the dis-

tribution of the data set where 'Cl' represents clustered, 'S-Cl' semi-clustered and 'R' randomly distributed. The third column displays the best solution found using the H_2 heuristic after a time limit of 1500 seconds. Figure 4.15 displays the average computational times taken for each value of n.

		H_2 Heuristic	Optimal Solution							
Dist	p	Z_1	Z^*	Loop CPU Time (secs) [a]	# Loops	Max Circles (secs)	CPLEX (secs)		Max Circles (%)	CPLEX (%)
Cl	25	56.871	53.677	1704.10	886	285.75	1245.46	1.42	16.77	73.09
Cl	50	38.082	32.660	27.95	27	3.71	12.84	0.42	13.28	45.95
Cl	75	31.827	24.279	26.12	60	2.93	16.31	0.24	11.23	62.46
Cl	100	25.480	19.963	18.87	79	1.64	13.60	0.14	8.70	72.08
Average	**63**	**38.065**	**32.645**	**444.26**	**263**	**73.51**	**322.08**	**0.56**	**12.50**	**63.40**
S-Cl	25	103.393	89.0983	2946.33	742	653.99	1891.80	2.54	22.20	64.21
S-Cl	50	72.890	53.8911	442.70	235	63.27	226.14	0.90	14.29	51.08
S-Cl	75	60.891	38.852	133.02	66	18.80	43.72	0.46	14.13	32.87
S-Cl	100	49.498	29.568	107.34	172	14.44	61.33	0.23	13.45	57.14
Average	**63**	**93.694**	**52.852**	**907.35**	**304**	**187.63**	**555.75**	**1.03**	**16.02**	**51.33**
R	25	119.474	107.353	157.67	16	13.10	57.49	6.43	8.31	36.46
R	50	82.619	67.151	53.81	56	4.42	35.14	0.58	8.19	65.32
R	75	63.389	51.573	14.57	32	1.15	9.50	0.27	7.92	65.21
R	100	54.722	40.140	14.10	61	1.00	9.65	0.13	7.08	68.45
Average	**63**	**80.051**	**66.554**	**60.04**	**41**	**4.92**	**27.95**	**1.85**	**7.88**	**58.86**

[a] This excludes computational time for the H_2 heuristic.

Table 4.16: Results for the generated data set where $n = 400$ using the Revised Drezner's Algorithm

Note that the generated data sets can be collected from the authors or accessed from the Centre for Logistics and Heuristic Optimisation (CLHO (2015)) website http://www.kent.ac.uk/kbs/research/research-centres/clho/datasets.html by following the links to 'continuous data sets' and then to 'random $p-$centre'.

Tables 4.16, 4.17 and 4.18 yield interesting results. The first table would suggest that clustered or semi-clustered data takes longer to solve on average than the randomly distributed data for the smallest data size ($n = 400$). Table 4.17 shows that the computational time begins to even out over the three distributions when the data size increases to $n = 600$. However, for the largest data set ($n = 800$), Table 4.18 shows a dramatic change. The clustered data now needs proportionally much less computational time to be solved compared to the randomly distributed data. Furthermore, the semi-clustered data also requires much less computational time compared to the randomly distributed data. Therefore these results suggest that as the size of the

		H_2 Heuristic	Optimal Solution							
Dist	p	Z_1	Z^*	Loop CPU Time (secs) [a]	# Loops	Max Circles (secs)	CPLEX (secs)		Max Circles (%)	CPLEX (%)
Cl	25	64.070	57.618	1977.52	42	185.65	447.96	10.73	9.39	22.65
Cl	50	44.003	36.739	1827.59	287	223.48	1077.57	4.49	12.23	58.96
Cl	75	34.436	25.986	1016.40	427	111.65	697.31	1.39	10.98	68.61
Cl	100	30.566	20.826	749.11	507	77.06	543.40	2.51	10.29	72.54
Average	**63**	**43.269**	**35.292**	**1392.66**	**316**	**149.46**	**691.56**	**4.48**	**10.72**	**55.69**
S-Cl	25	73.135	64.368	2269.07	43	157.70	496.03	14.70	6.95	21.86
S-Cl	50	51.587	41.904	872.52	70	71.70	316.32	14.16	8.22	36.25
S-Cl	75	41.135	31.275	1857.44	672	217.89	1328.04	2.69	11.73	71.50
S-Cl	100	35.203	24.323	494.76	257	52.83	315.74	0.92	10.68	63.82
Average	**63**	**50.273**	**40.468**	**1373.45**	**261**	**125.03**	**614.03**	**8.12**	**9.40**	**48.36**
R	25	74.402	67.979	1781.05	24	73.29	874.39	61.60	4.12	49.09
R	50	52.366	44.411	2917.39	26	15.09	2786.28	281.09	0.52	95.51
R	75	41.001	33.616	102.84	33	5.82	67.45	5.15	5.66	65.58
R	100	35.250	27.023	99.47	115	7.45	71.68	0.64	7.49	72.06
Average	**63**	**50.005**	**43.257**	**1225.18**	**50**	**25.41**	**949.95**	**87.12**	**4.45**	**70.56**

[a] This excludes computational time for the H_2 heuristic.

Table 4.17: Results for the generated data set where $n = 600$ using the Revised Drezner's Algorithm

data become larger, the more clustered that data set is the less computational time is required for the DEA to solve the problem optimally.

		H_2 Heuristic	Optimal Solution							
Dist	p	Z_1	Z^*	Loop CPU Time (secs) [a]	# Loops	Max Circles (secs)	CPLEX (secs)		Max Circles (%)	CPLEX (%)
Cl	25	63.776	60.879	3553.70	94	412.05	1641.80	16.89	11.59	46.12
Cl	50	48.089	40.265	9330.68	981	1108.46	6746.15	8.04	11.88	72.30
Cl	75	36.634	30.446	2420.53	564	240.32	1851.81	5.47	9.93	76.50
Cl	100	31.544	24.824	1748.96	711	161.83	1388.90	6.76	9.25	79.41
Average	**63**	**45.011**	**39.104**	**4263.47**	**588**	**480.67**	**2907.17**	**9.29**	**10.66**	**68.58**
S-Cl	25	90.593	81.238	22121.5	345	2472.25	11249.60	238.46	11.18	50.85
S-Cl	50	62.332	51.051	3405.79	73	190.46	1319.38	41.02	5.59	38.74
S-Cl	75	50.487	38.596	10793.60	1565	1266.10	8056.53	4.43	11.73	74.64
S-Cl	100	45.900	31.205	4682.38	1021	487.81	3387.57	9.42	10.42	72.35
Average	**63**	**62.328**	**50.523**	**10250.80**	**751**	**1104.16**	**6003.26**	**73.33**	**9.73**	**59.15**
R	25	126.385	115.444	15996.70	187	944.04	9461.35	55.79	5.90	59.15
R	50	89.059	75.935	79663.90	54	73.98	78929.40	10187.30	0.09	99.08
R	75	71.218	58.217	3940.31	36	20.00	3759.52	756.78	0.51	95.41
R	100	61.850	48.545	1032.94	104	22.43	924.72	121.00	2.17	89.52
Average	**63**	**87.128**	**74.535**	**25158.46**	**95**	**265.11**	**23268.75**	**2780.22**	**2.17**	**85.79**

[a] This excludes computational time for the H_2 heuristic.

Table 4.18: Results for the generated data set where $n = 800$ using the Revised Drezner's Algorithm

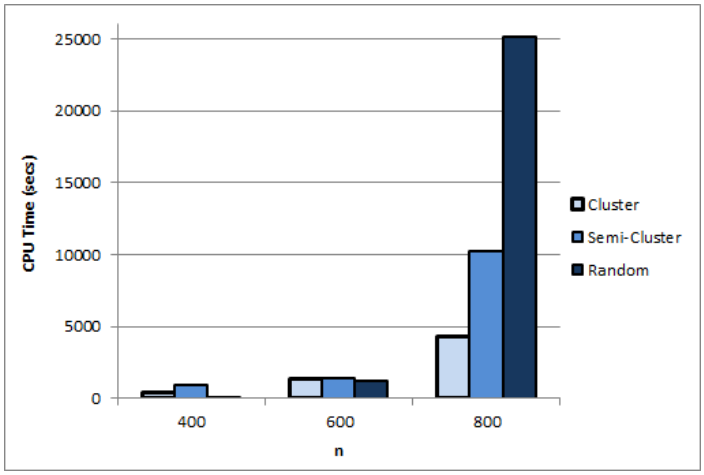


Figure 4.15: Average computational time for generated clustered, semi-clustered and randomly distributed data sets

4.10 Summary

This chapter has investigated an optimal algorithm proposed by Drezner (1984a) to solve the continuous $p-$centre problem. Opportunities to improve Drezner's algorithm were highlighted, four enhancements were discussed and tested, an adaptive CPLEX policy was introduced and an enhanced algorithm, DEA, was proposed. The DEA was tested on five existing TSP-Library data sets: $pr439$, $rat575$, $rat783$, $pr1002$ and $rl1323$ for $p = 10, \ldots, 100$ and on three new types of large data sets, namely randomly spread, semi-clustered and clustered where $n = 400, 600$ and 800. The results for the known data sets show the DEA gives a very significant decrease in computational time which sometimes reaches an average reduction of 96%, yielding an algorithm that is faster, more efficient and so can be used to solve the continuous $p-$centre problem for large data sets where several optimal solutions for large instances were obtained for the first time. The proposed algorithm also showed its superiority by producing optimal solutions for all the new generated data sets while requiring a reasonable amount of computational time especially for the clustered and semi-clustered data sets.

The next chapter will revisit another optimal algorithm that applies a different approach by using problem reduction and relaxation.

Chapter 5

Relaxation-Based Algorithms for the Continuous $P-$Centre Problem

5.1 Introduction

This chapter will analyse and investigate a well-known method called relaxation that is used to solve location problems, including the continuous $p-$centre problem. Section 5.2 will begin by describing the classic relaxation algorithm. It will then examine three algorithms that are developed from the classic relaxation problem, namely the improved classic relaxation, the reverse relaxation and the binary relaxation. The reverse relaxation algorithm is investigated further, and four enhancements are proposed and empirically tested using the TSP-Library data sets and the new generated data sets first discussed in the previous chapter. It is shown that each enhancement yields a faster and more powerful algorithm, thus leading to a powerful relaxation-based algorithm.

5.2 Relaxation Algorithms

The relaxation method breaks large problems down into smaller, but dependent, sub-problems to be solved successively until optimality is reached. Chen & Chen (2013: 39) stated that a problem must have the following two properties for the relaxation technique to be worth using.

1. There must be an algorithm capable of answering the question "is there a solution to the problem with value better than Z?", and finding such a solution if one exists.

2. The problem must be such that if there is no solution to the sub-problem with a value better than Z, then there can be no solution to the full problem with a value better than Z.

The classic relaxation algorithm was first suggested by Handler & Mirchandani (1979) and the main steps are summarised in Figure 5.1 (adapted from Chen & Chen (2009)). It begins with an upper bound of infinity and a small subset of demand points. The algorithm determines if a feasible solution can be found for the subset of demand points with a better solution value than the current upper bound. In other words, the minimum number of facilities required to cover all the demand points in the subset within the upper bound is calculated. If this number is $\leq p$, then there is a feasible solution for the subset with a solution value less than the current upper bound, and so feasibility for the full problem is checked. If the solution is feasible for the full problem, the upper bound is updated, otherwise another demand point is added to the subset and the process is repeated. If a feasible solution cannot be found for the subset of demand points with a solution value better than the current upper bound, then the current upper bound is taken as the optimal solution value for the entire problem.

1. Set the upper bound to be ∞.
2. Choose a random subset of demand points, Sub.
3. Determine whether Sub has a feasible solution, $feasible$, with a solution value smaller than the upper bound.
    a) If $feasible$ cannot be found, return the current upper bound as the solution value.
    b) If $feasible$ can be found, continue to Step 4.
4. Determine whether $feasible$ is feasible for the full problem.
    a) If it is, reset the upper bound as the solution value of $feasible$ and go back to Step 3.
    b) Otherwise, add another demand point to Sub and go back to Step 3.

Figure 5.1: Classic Relaxation based on Chen & Chen (2009: 1648)

Chen & Chen (2009) suggest two enhancements for the classic relaxation algorithm, which will be referred to as the improved classic relaxation algorithm. The first enhancement updates the upper bound more frequently, and the second adds k demand points at a time to the subset where $k > 1$. Furthermore, they also developed two other algorithms that were briefly described in Chapter 2. These two algorithms will be referred to as the 'reverse relaxation' and the 'binary relaxation'. For completeness, they are adapted from Chen & Chen (2009: 1649-1650) and described again here in

1. Set the lower bound to be 0.
2. Choose a random subset of demand points, Sub.
3. Determine whether Sub has a feasible solution, $feasible$, with a solution value smaller than the lower bound.
    a) If $feasible$ cannot be found, generate a new lower bound by finding the smallest radius of a covering circle created from Sub that is larger than the current lower bound and go back to Step 3.
    b) If $feasible$ can be found, continue to Step 4.
4. Determine whether $feasible$ is feasible for the full problem.
    a) If it is, halt and return $feasible$ as the final solution.
    b) Otherwise, add k demand points to Sub and go back to Step 3.

Figure 5.2: Reverse Relaxation based on Chen & Chen (2009: 1649)

Figures 5.2 and 5.3 respectively.

Although these algorithms are proven to be efficient at solving the discrete $p-$centre problem, they have not been tested for similar, large data sets as one may expect for the continuous case. For example, Chen & Chen (2009) only record one table of results for one instance only using the TSP-Library data set $pr439$ for $p = 10, \ldots, 100$ where they found exact solutions for all values of p. This result is promising, and highlights the potential that these type of algorithms may be used to solve larger, more complex problems if revisited.

After studying the computational times, the reverse relaxation algorithm was found to perform the best (i.e. fastest) for $p = 10-30$ while the binary relaxation algorithm performed the best for the remaining values of p (i.e. $p = 40-100$). However, there is little evidence to say which algorithm is the most efficient as the results are strictly based on this one data set. Furthermore, different and specific values of k worked best for each algorithm. Therefore, a further investigative work was carried out in this study to compare these two promising algorithms with an aim to identify the most promising algorithm that we can focus on for further investigations.

1. Set the lower bound (LB) to be 0 and the upper bound (UB) to be ∞.
2. Choose a random subset of demand points, Sub.
3. Set the coverage distance, CD, to be $\frac{LB+UB}{2}$.
4. Determine whether Sub has a feasible solution, $feasible$, with a solution value smaller than CD.
    a) If not, set $LB = CD$ and go back to Step 3.
    b) If $feasible$ can be found, continue to Step 5.
5. Determine whether $feasible$ is feasible for the full problem.
    a) If not, add k demand points to Sub and go back to Step 4.
    b) If so, save $feasible$ as *Best Candidate* found so far, and update UB to be the solution value of $feasible$, Z.
    If no solution less than Z can be found for Sub, halt and return *Best Candidate* as the final solution.

Figure 5.3: Binary Relaxation based on Chen & Chen (2009: 1650)

Initial Results

Chen & Chen's investigation was replicated by running both the reverse relaxation algorithm and the binary relaxation algorithm for the data set $pr439$ where $p = 10, \ldots, 100$ and $k = 1, \ldots, 10$. Table 5.1 shows Chen & Chen's first results for the binary and the reverse relaxation algorithm where $k = 6$ and 2 respectively. For comparison, our results are also shown for the same k values where Chen & Chen yielded their best results. Furthermore, the k value that required the smallest amount of computational time in our experiment is also reported. The best (i.e. fastest) results for the binary-based method were obtained when $k = 8$, and for the reverse-based method when $k = 3$. For clarity, the best results found for each value of p are also highlighted in bold.

When analysing the results, it becomes clear that Chen & Chen's results cannot be replicated easily, as we had different computational times and found different values of k to be more effective. This is because the algorithms proposed by Chen & Chen are not deterministic, and so the speed and the efficiency of the algorithm relies greatly on the initial subset of the demand points. This highlights the disadvantage of such algorithms, which we wish to examine in this chapter.

		Binary Relaxation			Reverse Relaxation		
		Chen & Chen's Results, $k^{\perp} = 6$	Original Implementation, $k^{\perp} = 6$	Original Implementation (best), $k^{\perp} = 8$	Chen & Chen's Results, $k^{\perp} = 2$	Original Implementation, $k^{\perp} = 2$	Original Implementation (Best), $k^{\perp} = 3$
p	Z^*	CPU Time (secs)	CPU Time (secs)	CPU Time (secs)	CPU Time (secs)	CPU Time (secs)	CPU Time (secs)
10	1716.510	4.53	3.07	9.30	**0.84**	2.41	4.18
20	1029.715	7.28	11.28	6.15	**2.63**	3.02	3.22
30	739.193	14.33	49.78	22.04	6.16	7.62	**3.75**
40	580.005	58.92	104.41	65.82	93.38	66.54	**38.69**
50	468.542	78.89	194.71	104.30	207.45	95.46	**63.62**
60	400.195	29.98	35.90	74.86	62.19	161.96	**24.49**
70	357.946	27.72	**25.77**	28.77	103.28	461.17	57.12
80	312.500	31.04	35.83	**21.51**	172.59	68.12	41.62
90	280.903	40.05	39.56	**33.45**	157.07	49.70	63.24
100	256.680	35.07	42.74	21.98	60.40	68.13	**17.32**
Average	**614.218**	**32.78**	**54.31**	**38.82**	**86.60**	**98.41**	**31.73**

$^{\perp}$ Best result for $k = 1, \ldots, 10$.

Table 5.1: Initial Results for the Binary and Reverse Relaxation Algorithms where $n = 439$

Furthermore, Chen & Chen's results also suggest that the binary relaxation algorithm is generally the faster of the two. Our results, however, suggest that both algorithms show efficiency, with the reverse relaxation algorithm showing a slightly faster rate on average. As we wish to focus on and enhance one algorithm, we suggest that the reverse relaxation algorithm shows marginally more potential in terms of finding the optimal solution. Therefore, this algorithm is chosen for further development.

It is important to note that the time complexity for both the binary and the reverse relaxation algorithm is bounded by Step 3 as this is NP-hard.

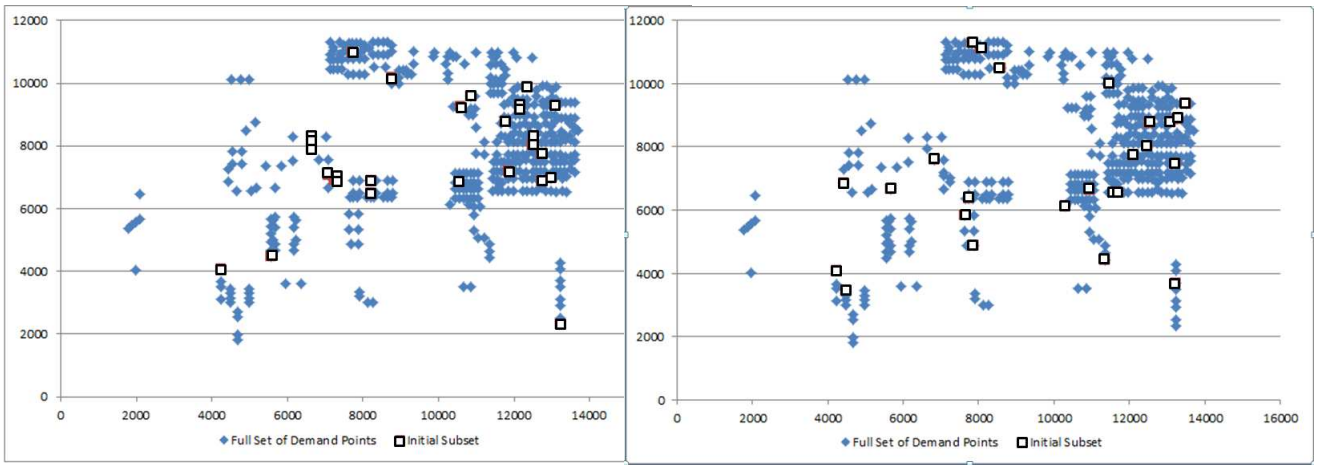
5.3 An Enhancement-Based Algorithm

The reverse relaxation algorithm aims to find an efficient small subset of demand points such that the exact solution to the full problem can be found. This can be achieved by finding a deterministic way to carefully select the initial subset of demand points, and efficiently adding k demand points to that subset. These ideas inspired the four enhancements which will be explained and investigated in this section. The first generates an initial subset deterministically, the second develops an efficient scheme for

adding k demand points to the subset, the third proposes a jumping scheme to save computational time and the fourth creates a dynamic scheme to determine the value of k.

5.3.1 A Deterministic Generator for the Initial Subset (Step 2 of Figure 5.2)

The computational time required for the reverse relaxation algorithm has a great dependence on the subset selected at the beginning. The results in the previous section suggest that the algorithm works efficiently and quickly; however this is not always the case. The results in Section 5.2 only give the best computational times, and had the algorithm only run once for one value of k (rather than having several different values to choose from) the results may have yielded different computational times.



(a) Run One (b) Run Two

Figure 5.4: Initial subset for $n = 439,\ p = 50,\ k = 5$

The variance of computational times can be easily demonstrated with a small example. The reverse relaxation algorithm was run twice to solve the data set $pr439$ where $p = 50$ and $k = 5$. In this particular experiment, run one found the optimal solution after 169.1 seconds, whereas the second run found it in less than half the time at 72.9 seconds. Figures 5.4a and 5.4b show the initial subset of demand points for the first run and the second run respectively. In Figure 5.4a, the demand points are more clustered, whereas during the second run (see Figure 5.4b) the initial subset of demand points were more evenly spread throughout the full data set. This therefore makes the initial subset given in the second run closer to the optimal solution than its counterpart

subset generated in run one.

This simple example demonstrates the initial motivation for the first enhancement for the reverse relaxation algorithm by highlighting that it may be preferable is to select an evenly spread out initial subset over the data's distribution in a deterministic way. This could lead to regaining computational time as the algorithm would be less sensitive to the initial subset of the demand points.

Generation of the initial subset of demand points

The first enhancement, which will be referred to as SubE1, proposes a scheme to find the minimum number of dispersed initial demand points. For clarity, let:
$d'_{i,l}$: the Euclidean distance from demand point i to demand point l;
Sub: the subset of demand points.

SubE1: An Overview

i) Firstly, we construct the convex hull encompassing all the demand points. Let CS be the set of points defining the vertices of the convex hull. In this study we used the quickhull algorithm proposed by Barber, Dobkin & Huhdanpaa (1996).

ii) The demand point $\hat{i} \in I$, that sits the greatest sum distance from all $i' \in CS$, is identified by

$$\hat{i} = Arg(\underset{i \in I \setminus CS}{Max}\{\sum_{i' \in CS} d'_{i,i'}\}), \tag{5.1}$$

and set $Sub = \{\hat{i}\}$.

iii) The algorithm then enters the third stage where

a) All $i \in I$ are allocated to their closest facility $j \in Sub$. This is represented by the allocation matrix Al.

$$Al_{i,j} = \begin{cases} 1 \text{ if } d_{i,j} \leq d_{i,j^*} \ \ \forall i \in I, \ \forall j, j^* \in Sub : j \neq j^*, \\ 0 \text{ else.} \end{cases}$$

b) The temporary facility with the most allocated demand points, j_{Max}, is

identified where

$$j_{Max} = Arg(\underset{j \in Sub}{Max}\{\sum_{i \in I} Al_{i,j}\}). \tag{5.2}$$

c) The demand point that is both allocated to j_{Max} and sits the furthest away from it, say i^*, is then identified as follows:

$$i^* = Arg(\underset{i \in I}{Max}(d_{i,j_{max}} : Al_{i,j_{Max}} = 1). \tag{5.3}$$

d) The demand point i^* is then added to the initial subset (i.e., $Sub = Sub \cup \{i^*\}$).

We will now discuss the method used to determine the cardinality of the initial subset of demand points.

Determining the minimum $|Sub|$

The minimum number of demand points, r, needed for the initial subset to yield an initial solution (i.e. p circles) is found by Chen & Handler (1987). They state that the number of circles that can be formed using r' demand points is derived from Equation (5.4).

$$r = \underset{r' \in \mathbb{N}}{Min} \ : \binom{r'}{3} + \binom{r'}{2} + r' \geq p. \tag{5.4}$$

To save computational time, the algorithm finds $(r-1)$ demand points using $SubE1$.

Note that r in Equation (5.4) represents the maximum number of circles made up from 1, 2 or 3 critical points first discussed in Section 2.2.4, Chapter 2.

The number of circles created from this initial subset of size r may not be large enough to form an initial solution (i.e. the number of circles $< p$), as some may be discarded due to the geometry of the critical demand points. If this is the case, the algorithm returns back to part (iii) and adds another demand point to the initial subset one by one until an initial solution can be found. The algorithm for SubE1 can be found in Figure 5.5.

1. Set $Sub = \emptyset$. Find the demand points $i' \in I$ that form the vertices of the convex hull, defined as the set CS.

2. Find the demand point $\hat{i} \in I \setminus CS$ that has the largest sum distance from all $i' \in CS$ using Equation (5.1). Set $Sub = Sub \cup \{\hat{i}\}$.

3. Determine the value of r using Equation (5.4).

4. While $|Sub| < r$ do:
   Allocate all $i \in I$ to their closest $j \in Sub$. Determine j_{Max} using Equation (5.2) and then find the furthest demand point $i^* \in I$ that is allocated to j_{Max} using Equation (5.3). Set $Sub = Sub \cup \{i^*\}$.

5. Find all circles made by one, two or three demand points from $i'' \in Sub$. If the number of circles is $\geq p$, return Sub as the initial subset and stop.
   Else, set $r = r + 1$ and go back to Step 4.

Figure 5.5: Initial Subset Algorithm (SubE1)

This method also allows the distribution of the data to be considered by allocating all demand points to their closest $j \in Sub$ and noting the one that has the most allocated to it. This means the initial subset does not only yield a well dispersed set of demand points, but additionally it is proportionally dispersed over clustered areas and evenly spread areas. This gives a more efficient subset of demand points to begin the algorithm. Furthermore, by checking to see if the subset can produce an initial solution each time another demand point is added, the size of the subset is minimised and so reduces redundant calculations.

Illustrative Example

Figure 5.6 shows the demand points (diamonds) for the data $pr439$. Let $p = 30$, and four facilities, F_1, F_2, F_3 and F_4 have already been found using SubE1. Equation (5.4) states that as $p = 30$ then $r = 6$, and so (at least) two more demand points must be added to the initial subset. If the furthest point from its allocated facility was taken, this would mean that point P_1 would be selected next for Sub as it sits the furthest from its allocated facility, F_1, than all other demand points from their allocated facilities. However, even though this is an extreme point, this is not the best point to select as it lies in an area where demand points are sparse.

After studying the distribution, it is clear to see the best place to select a point is

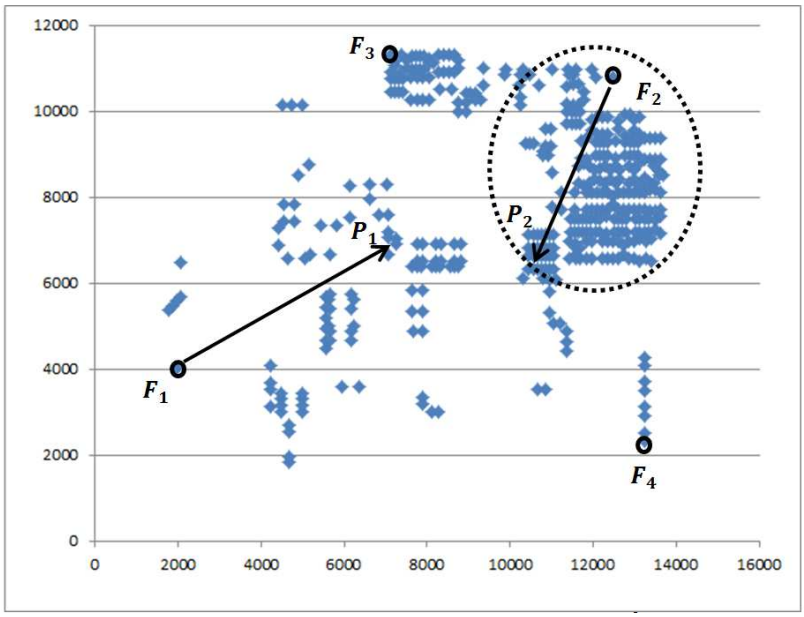


Figure 5.6: Illustrative example of SubE1

inside the dashed circle area, as this is a highly populated area with only one facility. Therefore, by taking into account both a) the distance from a demand point to a facility and b) the number of demand points a facility serves, SubE1 would select point P_2 as the next facility. This creates a more evenly spread selection of facilities over the data's distribution.

Figure 5.7 shows the initial subset that has been found for the data set $pr439$ where $p = 50$ using SubE1. Note that $|Sub| = 24$. The Initial Subset Algorithm would proceed to find the extreme demand points that form the convex hull, and the point that has the greatest sum distance from these extreme points. The third stage would then find another six points straight away. This is due to $\binom{6}{3} + \binom{6}{2} + 6 = 41$ ($< p$) and $\binom{7}{3} + \binom{7}{2} + 7 = 63$ ($> p$), thus making $r = 7$. The third stage would then continue to find the remaining 17 points one by one until an initial solution is obtained.

For comparison, Figure 5.8 shows the critical points of the circles for the optimal solution for $pr439$ where $p = 50$. Note that the number of critical points is 88. When comparing Figures 5.7 and 5.8, it can be seen that there are similarities as the demand points tend to be situated at extreme parts of clusters (in the corners or near the edge). Therefore, it suggests that enhancement one creates a good initial subset in a deterministic way.

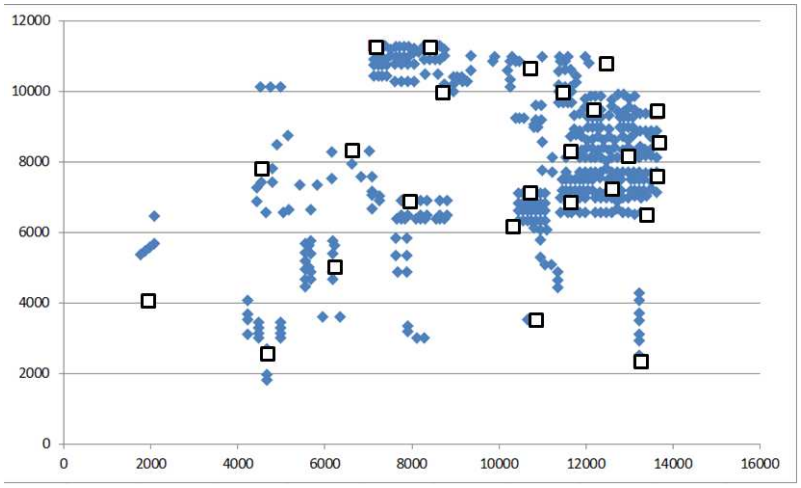


Figure 5.7: Initial subset for $n = 439$, $p = 50$ using SubE1

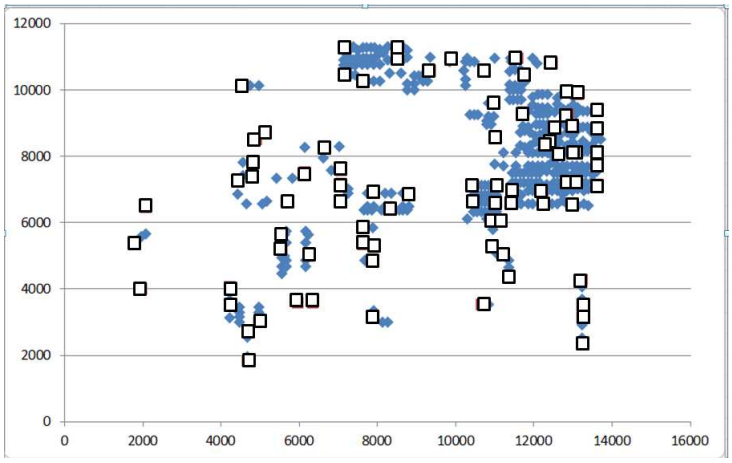


Figure 5.8: Critical points for the optimal solution for $n = 439$, $p = 50$

Why all $i' \in CS$ are not necessarily added to the initial subset

Observation 1. *Take two points $i_1, i_3 \notin CS$ and point $i_2 \in CS$. If the angle $\angle i_1 i_2 i_3 > 90°$, then the point i_2 can be encompassed by the covering circle formed from i_1 and i_3.*

This observation is illustrated in Figure 5.9 with an example where the point i_2 is encompassed by a covering circle formed from the two points $i_1, i_3 \notin CS$.

As the third part of SubE1 is designed to find the dispersed, extreme points in the plane it might seem reasonable to add all $i \in CS$ to Sub automatically. However, observation 1 demonstrates that the extreme points that form the convex hull are not necessarily critical points for their covering circles. Although SubE1 could select some points that form the vertices of the convex hull to be in Sub, it is also designed to find a good initial subset based on the data distribution and thus be more selective about the

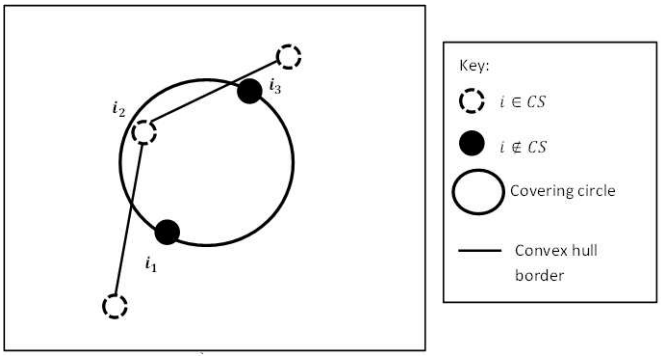


Figure 5.9: Observation 1

points that are chosen. Therefore, instead of automatically adding all $i \in CS$ to Sub, we acknowledge that this may increase the problem size unnecessarily and so allow the third stage (part (iii)) to choose the most appropriate points.

Results using SubE1

Table 5.2 shows the comparison between the computational times for Chen & Chen's results, our first results (found in Section 5.2) and the results with SubE1 added to the reverse relaxation algorithm for the TSP-Library data set $pr439$.

		CPU Times (secs)				
		Chen & Chen's Results	Our First Results			Our Results With SubE1
p	Z^*	Best$^{\perp}$ ($k=2$)	Best$^{\perp}$ ($k=3$)	Worst ($k=1$)	Average ($k=1,\ldots,10$)	Best$^{\perp}$ ($k=2$)
10	1716.510	0.84	4.18	2.27	29.65	1.98
20	1029.715	2.63	3.22	2.31	18.94	2.12
30	739.193	6.16	3.75	5.05	27.10	42.39
40	580.005	93.38	38.69	29.29	57.96	38.96
50	468.542	207.45	63.62	69.20	127.83	67.84
60	400.195	62.19	24.49	77.55	54.59	14.18
70	357.946	103.28	57.12	52.18	107.24	45.33
80	312.500	172.59	41.62	620.60	188.09	52.39
90	280.903	157.07	63.24	274.08	85.05	85.45
100	256.680	60.40	17.32	119.36	51.92	31.59
Average	**614.22**	**86.60**	**31.73**	**125.18**	**74.81**	**38.22**

$^{\perp}$ Best result for $k = 1, \ldots, 10$.

Table 5.2: Results comparing the Reverse Relaxation Algorithm with and without SubE1

These results may suggest initially that the first enhancement does not improve the computational time when compared with our original results where $k = 3$. However, as stated before, SubE1 has the advantage of being deterministic. When compared with

Chen & Chen's results, it can be seen that the reverse relaxation algorithm including SubE1 required less computational time. Due to the original algorithm being non-deterministic, some runs may be much faster than others. Therefore, it is promising to see the computational time taken by the algorithm including SubE1 is much closer to that of a 'good' run for the original algorithm. In addition, a deterministic selection method for the initial subset, such as SubE1, is more reliable as it can be replicated by other researchers while not requiring extremely large computational times that a random subset might incur.

5.3.2 An Efficient Scheme for Adding Demand Points (Step 4b in Figure 5.2)

Chen & Chen's reverse relaxation algorithm states that if more demand points are to be added to the subset, the k furthest ones from their allocated facilities are chosen. Though this is a good idea, this method could be improved further by identifying its weakness through the following 'worst-case scenario' example.

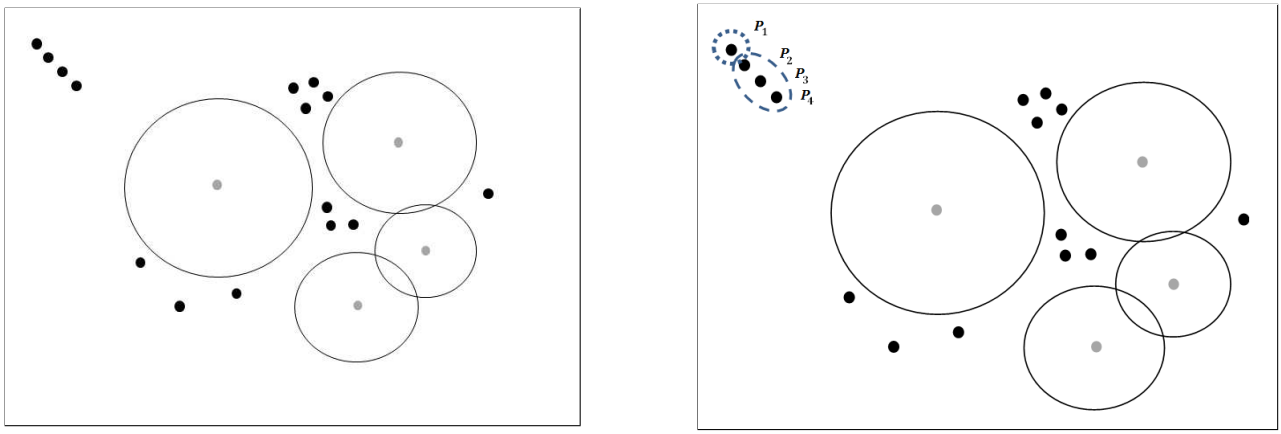
In order to explain the second enhancement, we recall from Chapter 4 that $R(K)$ is the radius of the smallest circle encompassing all points in K ($K \subset I$).

An Illustrative Example

Figure 5.10a demonstrates a solution for the relaxed subset where $p = 4$. The grey dots are the facility locations (circle centres) for the subset's solution. The black dots are the uncovered demand points in the full problem. It is clear to see this solution is not feasible for the full problem.

Let us say $k = 4$. According to the original algorithm, all uncovered demand points must now be allocated to their nearest facility and the k furthest points will be added to the subset. In this example, the first demand point that will be allocated to the subset, P_1, is the one encompassed by its own dotted circle and labelled P_1 in Figure 5.10b. Adding P_1 to the subset could contribute to a good result as it would guide the algorithm towards the optimal solution.

However, the other three demand points that would be added to the subset, labelled



(a) (b)

Figure 5.10: Adding k furthest demand points example

P_2, P_3 and P_4 are the three demand points that are circled with a dashed line in Figure 5.10b. The addition of these three demand points to the subset does not contribute to improving the solution as the new information is now redundant. This is because $R(K \cup \{P_1\}) \geq R(K \cup \{P_s\}), s = 2, 3, 4$, as P_1 lies further from circle C_j. Adding P_2, P_3 or P_4 to Sub would mean there were extra calculations that are both redundant and time consuming. This claim is stated in Lemma 4.

Lemma 4. *Take two demand points, i^1 and i^2, that are not encompassed by C_j. If $d_{i^1,j} > d_{i^2,j}$, then $R(K \cup \{i^1, i^2\}) > R(K \cup \{i^2\})$.*

Proof.

$$R(K \cup \{i^1, i^2\}) = \frac{r_j + d_{i^2,j} + (d_{i^1,j} - \nu)}{2} - \epsilon$$

where $0 \leq \nu \leq d_{i^2,j}$, and

$$R(K \cup \{i^2\}) = \frac{r_j + d_{i^2,j}}{2} - \epsilon.$$

If $R(K \cup \{i^1, i^2\}) > R(K \cup \{i^2\})$

$$\implies \frac{r_j + d_{i^2,j} + (d_{i^1,j} - \nu)}{2} - \epsilon > \frac{r_j + d_{i^2,j}}{2} - \epsilon,$$

$$\implies \frac{d_{i^1,j} - \nu}{2} > 0,$$

$$\implies d_{i^1,j} > d_{i^2,j}.$$

□

Enhancement two, which will be referred to as AddE2, aims to overcome this shortcoming by slightly altering this selection rule. In order to explain AddE2, the following definition is given.

Definition 5.3.1. *An* **artificial circle** *is a circle whose radius, r_j, has been enlarged to Z_t, where Z_t is the solution value at iteration t.*

At any iteration t, all circles' radii are artificially increased to size Z_t and the solution is analysed again to see what demand points remain uncovered. This allows more demand points to be covered whilst not compromising the solution quality.

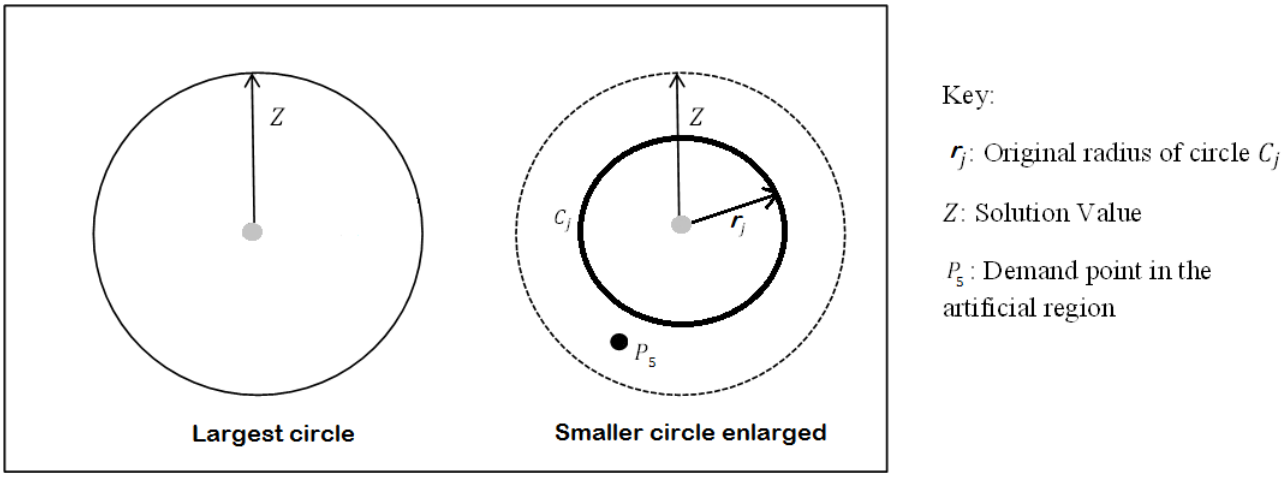


Figure 5.11: Construction of an artificial circle with radius Z

Figure 5.11 demonstrates the usefulness of artificial circles. The demand point P_5 appears to be uncovered by a solution circle. However, once the smaller solution circle is enlarged into an artificial circle (the dashed circle) by increasing its radius r_j to Z, it then becomes clear that point P_5 is now covered with the solution value Z.

AddE2 uses artificial circles to help enhance and improve the selection technique when choosing the new demand points to be added to the subset in the following two steps.

1. For iteration t, increase all circle's radius to Z_t. Only the demand points that are not covered by an artificial circle can now be selected for the subset.

2. The uncovered points are then allocated to their closest facility creating at most p clusters of demand points. However, we suggest adding the k furthest demand

points but with the added condition that *only one point is selected from each cluster*. This strategy therefore allows a wider distribution of demand points to be selected and avoids the weakness highlighted in Figure 5.10b from occurring.

The detailed algorithm for AddE2 is outlined in Figure 5.12

1. Input: set of solution circles (facilities) F and value k.
2. Allocate all the demand points $i \in I$ to their closest facility $j \in F$ and define the allocation matrix $A_{i,j}$ where $A_{i,j} = 1$ if demand point i is allocated to facility j, else $A_{i,j} = 0$. Record $Z = Max(r_j : \ j \in F)$.
3. Artificially increase all r_j so that $r_j = Z \ \forall \ j \in F$.
4. Update the allocation matrix such that $A_{i,j} = 1$ if demand point i is both allocated to facility j and $d_{i,j} > Z$.
5. Find the k furthest demand points from their allocated facility such that only one demand point is selected from each facility.

Figure 5.12: Point Selection Algorithm (AddE2)

Computational Results

AddE2 was first added on its own to the reverse relaxation algorithm and tested on the data set $pr439$ where $p = 10, \ldots, 100$ and $k = 1, \ldots, 10$. These results are given in column 4 of Table 5.3, and show that the algorithm has improved with an average decrease in computational time of 10.6%.

Although this is an encouraging result, due to the fact that the algorithm is not deterministic, fast results such as these cannot be replicated with AddE2 alone. To achieve the desired outcome, both SubE1 and AddE2 are incorporated into the reverse relaxation algorithm. The results, shown in the last column Table 5.3, show that the deterministic algorithm with these two enhancements yields the optimal solution in a faster time with an average decrease of computational time of 11.3%.

		CPU Time (secs)		
p	Z^*	Without Enhancements$^{\perp}$, $k = 3$	With AddE2$^{\perp}$, $k = 2$	With SubE1 & AddE2$^{\perp}$, $k = 3$
10	1716.510	4.18	2.08	1.81
20	1029.715	3.22	4.63	2.32
30	739.193	3.70	10.51	5.94
40	580.005	38.69	24.87	49.97
50	468.542	63.62	83.76	53.08
60	400.195	24.49	20.76	12.10
70	357.946	57.12	51.32	47.07
80	312.500	41.62	29.32	47.43
90	280.903	63.24	38.33	48.20
100	256.680	17.32	17.81	13.36
Average	**614.22**	**31.73**	**28.34**	**28.13**

$^{\perp}$ Best result for $k = 1, \ldots, 10$.

Table 5.3: Results for the Reverse Relaxation Algorithm without enhancements, with AddE2 and with SubE1 & AddE2 where $n = 439$

5.3.3 Jump-Based Lower Bound Update (Step 3a in Figure 5.2)

The reverse relaxation algorithm is constantly updating the lower bound resulting in a large number of iterations. Each new update, especially at the start of the search, produces a small increase from one iteration to the next making most 'jumps' to the next lower bound redundant. This section takes this drawback into account and attempts to provide a more efficient updating scheme.

Enhancement three, which will be referred to as JumpE3, aims to reduce the number of iterations by redefining what the next lower bound will be set to. Instead of being set as the smallest radius larger than the current lower bound, it will be set to the $jump^{th}$ smallest radius.

In the first experiment, we set $jump = 2$ which is displayed in Figure 5.13. By missing out the next lower bound (i.e. jumping to the second smallest radius larger than the current lower bound), the algorithm converges towards the optimal solution faster. A similar approach was first suggested for the discrete $p-$centre problem by Al-Khedhairi & Salhi (2005).

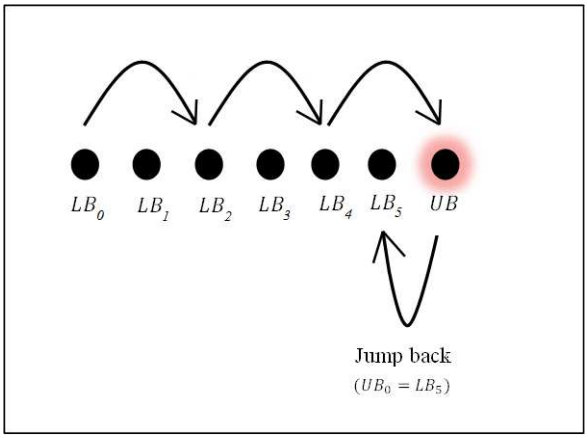


Figure 5.13: Checking which bound is optimal ($jump = 2$)

However, this means that an extra step is needed once the algorithm finds a solution that is feasible for the full problem, see Figure 5.13. At each step, the lower bound, LB, is updated to the second smallest radius larger then the current lower bound. Once a feasible solution is found, this becomes an upper bound, UB, for the optimal solution. However, the previously missed bound must now be checked to see whether or not this yields a feasible solution. This is shown as the 'jump back' step in Figure 5.13. If the missed value does yield a feasible solution, then it is is taken as the optimal solution, otherwise the current upper bound UB is the optimal solution value

JumpE3 was added to the enhanced algorithm and tested on the data set $pr439$ for $p = 10, \ldots, 100$ in steps of 10. The results are given in Table 5.4 which is organised in the following way. The first column shows the number of facilities located, p, and the second column shows the optimal solution value, Z^*. The next three columns show the number of iterations needed, the number of lower bound updates required and the total amount of computational time in seconds for the reverse relaxation algorithm with the enhancements SubE1 and AddE2 to reach optimality. The final three columns display the same information, but this time corresponding to the reverse relaxation algorithm with enhancements SubE1, AddE2 and JumpE3 where $jump = 2$.

The results show that the computational time required to find the optimal solution has significantly decreased. This gives a good indication that JumpE3 has the desired effect of decreasing the total number of iterations and the number of times the lower bound was updated. This is confirmed by comparing the information given in Columns

p	Z^*	With SubE1 & AddE2$^{\perp}$, $k=3$			With SubE1 & AddE2 & JumpE3$^{\perp}$, $k=4$		
		# Iterations	# LB Updates	CPU Time (secs)	# Iterations	# LB Updates	CPU Time (secs)
10	1716.510	272	224	1.80	202	140	1.05
20	1029.715	253	187	2.33	260	167	2.26
30	739.193	338	237	5.94	309	172	3.81
40	580.005	727	573	49.97	604	400	19.97
50	468.542	616	462	53.08	664	449	58.53
60	400.195	407	231	12.10	402	173	7.58
70	357.946	687	490	47.07	599	342	18.54
80	312.500	725	512	47.43	652	368	30.49
90	280.903	555	335	48.20	555	259	24.13
100	256.680	345	137	13.36	393	117	11.018
Average	**614.22**	**493**	**339**	**28.13**	**464**	**259**	**17.75**

$^{\perp}$ Best result for $k = 1, \ldots, 10$.

Table 5.4: Results for the Reverse Relaxation Algorithm with SubE1 & AddE2 with and without JumpE3, where $n = 439$ and $jump = 2$

3 & 4 with that of Columns 6 & 7 respectively.

Defining a critical and dynamically changing jump

Although jumping to the second lower bound is effective, this idea can be developed further. As the algorithm has many iterations until the optimal solution is finally reached by the lower bound, why not go to the 3^{rd} lowest bound, or the 4^{th}? In other words, Al-Khedhairi & Salhi set the new lower bound $LB_0 = LB_2$ where

$$LB_1 = min\{d_{i,j} : d_{i,j} > LB_0 \ \forall \ i \in I, j \in J\} \tag{5.5}$$

$$LB_2 = min\{d_{i,j} : d_{i,j} > LB_1 \ \forall \ i \in I, j \in J\}. \tag{5.6}$$

We aim to jump to the $jump^{th}$ lowest bound, such that $LB_0 = LB_{jump}$ where

$$LB_{jump} = min\{d_{i,j} : d_{i,j} > LB_{jump-1} \ \forall \ i \in I, j \in J\}. \tag{5.7}$$

Simple Comparison of jump = 2 vs. jump = 5

As an example, the computational time saved using JumpE3 when $jump = 2$ & 5 can be summarised below, where δ is the number of iterations needed to find the optimal solution when $jump = 1$, and $No_q^{(k)}$ is the number of iterations that can be saved when $jump = k$.

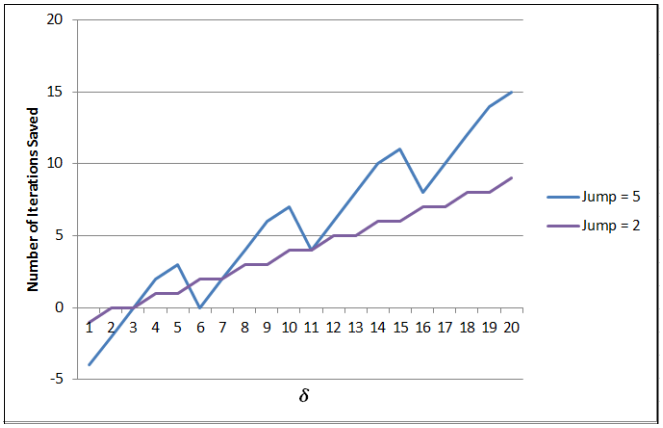


Figure 5.14: Iterations saved for $jump = 2, 5$ where $\delta = 1, \ldots, 15$

If $jump = 2$,

$$No_q^{(2)} = \begin{cases} \frac{\delta}{2} + 1 & \text{if } \delta \text{ is even,} \\ \frac{\delta+1}{2} + 1 & \text{if } \delta \text{ is odd.} \end{cases} \tag{5.8}$$

If $jump = 5$,

$$No_q^{(5)} = \begin{cases} \delta - (\frac{\delta}{5} + 1) & \text{if } q = 0, \\ \delta - (\frac{\delta - q}{5} + (6 - q)) & \text{if } q > 0, \end{cases} \tag{5.9}$$

where $\delta = q \bmod jump$.

The computational savings can be easily compared with a simple example. If $\delta = 100$, the total number of iterations saved when $jump = 2$ is $\frac{100}{2} + 1 = 51$, yielding a saving of 51%. This can be compared to when $jump = 5$, where the number of iterations saved is $100 - \frac{100}{5} + 1 = 79$, leading to a saving of 79%.

Figure 5.14 compares the number of iterations saved when $\delta = 1, \ldots, 20$ for $jump = 2$ and $jump = 5$. If $\delta < 3$ the solution is always found with less iterations when $jump = 2$, and if $\delta > 11$ the solution is always found with less iterations when $jump = 5$. If $\delta \geq 7$ and $jump = 5$, the number of iterations saved is either greater than or the same compared to $jump = 2$. Note that when $\delta = 6, 11$, there is a drop in the number of iterations saved for $jump = 5$ compared to $jump = 2$.

Defining the backward jump

Note that if an upper bound is found when $jump = 2$, the next lower bound to con-

sider if the one before which is the only value missed. However, if $jump > 2$, there will be more than one value missed leading to several options for choosing the next lower bound.

One option is to use a binary (bisector) method by taking the missed value that is halfway between the current lower bound and upper bound and continuing from there. Another way would be to return to the very first missed lower bound and continue from this point successively until the optimal value is reached.

However, in this study we apply a backward strategy where the idea is to start from the last lower bound that was missed. If this value is found to be an upper bound, we then proceed to the second last value that was missed and so on until a lower bound is found. Once a lower bound is found, the upper bound checked previously is therefore the optimal value.

Figure 5.15 illustrates the 'jump-back' method by showing an example where $jump = 5$. The algorithm first goes to LB_5, which yields a value that is not feasible for the full problem. It then proceeds to the next lower bound which is LB_{10}. This yields a feasible solution for the full problem, thus becoming an upper bound (i.e. $UB = LB_{10}$). The missed lower bound values (between LB_5 and LB_{10}) must now be checked to determine the optimal solution value. We begin by checking the last missed lower bound and working backwards from this point. Therefore, the first to be checked is LB_9. If this is infeasible (i.e., it is a lower bound), the value found at LB_{10} (i.e. UB) is optimal, else UB becomes LB_9 and we check LB_8 and so on.

Computational Results

The data set $pr439$ was run with the enhancements SubE1, AddE2 and JumpE3 where $jump = 5$. In other words, the next lower bound taken is the fifth lowest radius above the current lower bound. The results showed another decrease in the overall computational time for all values of k, with the fastest having an average of 10.20 seconds ($k = 4$). The full results for $k = 4$ can be found in Table 5.5.

After studying the results, several interesting observations are highlighted. Firstly,

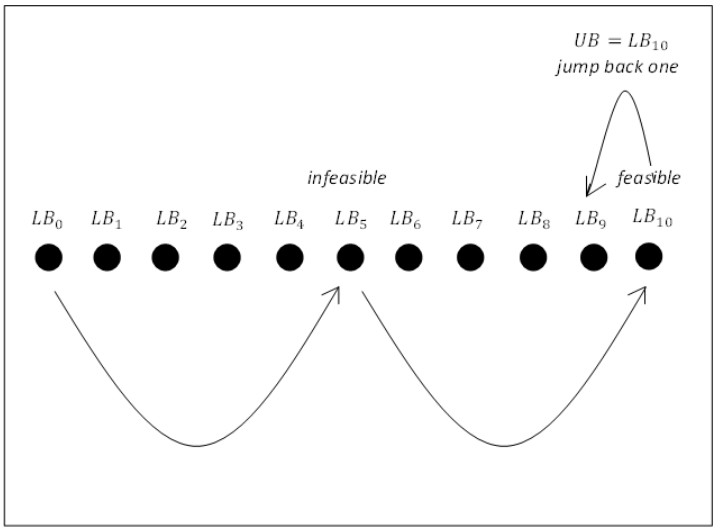


Figure 5.15: Jumping back demonstration where $jump = 5$

p	Z^*	$jump = 2$			$jump = 5$		
		# Iterations	# LB Updates	CPU Time (secs)	# Iterations	# LB Updates	CPU Time (secs)
10	1716.510	202	140	1.05	110	30	0.61
20	1029.715	260	167	2.26	153	52	1.03
30	739.193	309	172	3.81	210	68	3.18
40	580.005	604	400	19.97	372	160	12.09
50	468.542	664	449	58.53	382	165	20.13
60	400.195	402	173	7.58	297	64	10.22
70	357.946	599	342	18.54	386	131	9.27
80	312.500	652	368	30.49	412	34	20.36
90	280.903	555	259	24.13	390	91	15.44
100	256.680	393	117	11.02	337	40	9.63
Average	**614.22**	**464**	**259**	**17.75**	**305**	**94**	**10.20**

$^{\perp}$ Best result for $k = 1, \ldots, 10$.

Table 5.5: Results for the Reverse Relaxation Algorithm with SubE1, AddE2, JumpE3$^{\perp}$ where $n = 439$, $k = 4$, $jump = 2$ and $jump = 5$

even though the computational time is shorter, the average number of iterations is still larger than expected according to Equation (5.9). For example, originally for $p = 50$, $k = 4$, it took 1022 iterations to find the optimal solution. According to Equation (5.9), if $jump = 5$, the number of iterations that should be saved is 814. This would therefore allow the optimal solution to be found after 208 iterations. However, Table 5.5 states that 382 iterations were needed. This is because demand points are added to the relaxed subset, and so more circles are generated that have radius r_j such that $LB_{5(t-1)} < r_j < LB_{5t}$ for iteration t. Therefore, instead of having only four missed values, there are more circles (and so lower bound values) to check through once an upper bound has been established.

For this reason, a balance must be found between making a $jump$ as large as possible whilst minimising the number of circles to check through once an upper bound has been established. This issue will be addressed in the next part of this section.

A dynamic jumping scheme

This scheme aims to get the best of both worlds by making $jump$ large at the beginning so that redundant lower bounds can be checked through quickly, whilst decreasing $jump$ as the algorithm gets closer to the optimal solution and thus tightening the gap between the current lower bound and the first upper bound found. This variation of JumpE3 will be referred to as DyJumpE3.

This is a non increasing but discontinuous function of iterations $jump(t)$ with t denoting the t^{th} iteration, $jump(t) \in \mathbb{N}$. As the value of $jump$ will change throughout the algorithm, the computational time savings ($No_q^{(jump)}$) have been generalised for any value of $jump$.

$$No_q^{(jump)} = \begin{cases} \delta - [\frac{\delta}{jump} - 1] & \text{if } q = 0, \\ \delta - [\frac{\delta - q}{jump} + 1 + (jump - q)] & \text{if } q > 0. \end{cases} \tag{5.10}$$

The maximum number of jumps, $jump_{Max}$, must be chosen. In other words, the maximum number of lower bounds that are missed in an iteration. This is so the scheme has a range of integer numbers to choose from where

$$2 \leq jump(t) \leq jump_{Max} \quad \forall t.$$

Initially $jump = jump_{Max}$, and the scheme uses the proportion of uncovered demand points to decrease the value of $jump$. Let N_u^t be the number of uncovered demand points at iteration t. Using the proportion of demand points not covered at iteration t compared to the total number of demand points, the value of $jump$ can be determined

from its range using Equation (5.11).

$$jump(t) = \lfloor jump_{Max} - (jump_{max} \times (1 - \frac{N_u^t}{N}) + 0.5 \rfloor. \tag{5.11}$$

This scheme was tested on the data set $pr439$ where $jump_{Max} = 10$ and $k = 1, \ldots, 10$. The results are given in Table 5.6, where the best result is found when $k = 4$ with an average of 6.40 seconds. The worst computational time recorded was when $k = 1$, where the average required amount of computational time was recorded as 16.95 seconds. Therefore, we can claim that even the worst scenario has improved the computational time significantly. This suggests that the dynamic scheme delivers very promising results, and is worthwhile incorporating into the enhanced algorithm.

p	Z^*	# Iterations	# LB Updates	CPU Time (secs)
10	1716.510	147	21	0.56
20	1029.715	234	22	0.97
30	739.193	353	56	2.47
40	580.005	549	123	10.75
50	468.542	549	118	12.13
60	400.195	524	51	5.60
70	357.946	617	102	8.03
80	312.500	680	109	8.92
90	280.903	688	78	8.49
100	256.680	627	36	6.04
Average	**614.22**	**497**	**72**	**6.40**

Table 5.6: Results for the Reverse Relaxation Algorithm with SubE1, AddE2 & DyJumpE3$^{\perp}$ where $k = 4$

5.3.4 A Dynamic Scheme for the Determination k (Step 4b in Figure 5.2)

When the enhanced algorithm did not include DyJumpE3, the best results were often determined when $k = 2$ or 3. However, in Table 5.5 the best computational times were found when $k = 4$. This suggests that the value of k is very influential and critical to the success of the algorithm.

One way to identify the most appropriate estimator for k would be to perform a thorough statistical analysis. However, this requires relying on performing such exper-

iments with the expectation that a similar performance will remain valid when tested on new data sets. Another approach, which we think to be more robust, is to develop a self−learning method that will learn as the search progresses while incorporating the characteristic of the $p-$centre problem. Our final enhancement, PointE4, uses a self−learning method to determine the value of k during the algorithm. This scheme is developed using very similar ideas to the dynamic jumping scheme described previously.

It may seem unnatural to add a fixed number of demand points to the subset in each iteration, as the number of demand points that are uncovered is constantly decreasing. Therefore, by setting k to be a certain percentage of uncovered demand points, a proportional amount of demand points can be added to the subset. As the subset gets larger (and so the algorithm begins to slow) the number of demand points added to the subset gets smaller and hence the algorithm stays at a relatively good speed. It also means more points are added to the subset at the beginning, thus finding many necessary demand points quickly. Our fourth enhancement, PointE4, incorporates this idea to find a balance between adding more demand points at the beginning and decreasing the value of k as the algorithm starts to slow.

Previous results show that the reverse relaxation algorithm generally works best when $k > 1$. Therefore, at iteration t we set k to be

$$k(t) = Min(p, Max(2, \lfloor (0.02 \times N_u^t) + 0.5 \rfloor)). \tag{5.12}$$

In other words, the minimum value of k is 2 while being bounded by a maximum of p.

Incorporating all Four Enhancements

Table 5.7 shows the final results for $pr439$ with all four enhancements combined. For comparison, the computational times from the original implementation (Chen & Chen's and ours) are also shown. For simplicity, we did not record the detailed results for each enhancement as performed in the previous chapter, though more detailed computations could be added here.

It is clear that the algorithm with all four enhancements incorporated into it finds

		Original CPU Time			Reverse Relaxation		
		Chen & Chen's ($k = 2$)	Original Implementation ($k = 2$)	Original Implementation ($k = 3$)	With SubE1 & AddE2 & DyJumpE3 & PointE4		
p	Z^*	CPU Time (secs)	CPU Time (secs)	CPU Time (secs)	# Iterations	# LB Updates	CPU Time (secs)
10	1716.510	0.84	2.41	4.18	40	11	0.78
20	1029.715	2.63	3.02	3.22	53	9	1.48
30	739.193	6.16	7.62	3.75	68	9	2.86
40	580.005	93.38	66.54	38.69	93	13	9.97
50	468.542	207.45	95.46	63.62	107	26	13.79
60	400.195	62.19	161.96	24.49	99	12	10.00
70	357.946	103.28	461.17	57.12	130	25	16.77
80	312.500	172.59	68.12	41.62	126	19	14.95
90	280.903	157.07	49.70	63.24	139	17	20.83
100	256.680	60.40	68.13	17.32	128	7	15.55
Average	**614.220**	**86.60**	**98.41**	**31.73**	**98**	**15**	**10.70**

Table 5.7: Results for the Reverse Relaxation Algorithm with SubE1, AddE2, DyJumpE3 & PointE4 ($n = 439$)

the optimal solution in a faster computational time compared to Chen & Chen. These results are very encouraging as it means the developed, deterministic algorithm is more efficient and faster than the non-deterministic algorithm.

Furthermore, when comparing the results to Table 5.6, we can see the new average computational time is slightly higher with PointE4. However, it is worth pointing out that in this variant we do not rely on the given value k, but instead let it take its value from Equation (5.12) making the method more stable. One other aspect worth mentioning is that there is a significant decrease in the average number of iterations when using Point E4. This result is also useful, as it is very desirable to minimise the number of iterations as each one may require larger computational times, especially when solving in larger instances which will be shown next.

5.4 The Enhanced Reverse Relaxation Algorithm & Overall Results

Figure 5.16 describes the enhanced algorithm that incorporates all four enhancements discussed in this chapter. This algorithm will now be referred to as the Enhanced Reverse Relaxation Algorithm ($ERRA$).

1. Set the lower bound $LB = 0$, $c = 1$, $t = 0$ and $jump_{Max} = 10$.
2. Select the initial subset of demand points using Initial Subset Algorithm given in Figure 5.5.
3. Set $t = t + 1$. Determine if Sub has a feasible solution for the $p-$centre problem, $feasible$, with a solution value, $Z_t \leq LB$.
    a) If $feasible$ cannot be found, determine the value of $jump$ using Equation (5.11) and find the $jump^{th}$ smallest lower bound, LB_{jump}, that is larger than LB. Set $LB = LB_{jump}$ and go back to the beginning of Step 3.
    b) If $feasible$ can be found, find the number of uncovered demand points, N_u^t, and continue to Step 4.
4. Determine whether $feasible$ is feasible for the full problem.
    a) If it is, determine whether Sub has a feasible solution, $feasible$, with a solution value $\leq LB_{(jump-c)}$.
    If not, return $LB_{jump-(c-1)}$ as the final solution value.
    Else set $c = c + 1$ and repeat Step 4(a).
    b) Determine the value of k using Equation (5.12). Add k demand points to Sub using the Point Selection Algorithm in Figure 5.12 and go to Step 3.

Figure 5.16: The Enhanced Reverse Relaxation Algorithm ($ERRA$)

The $ERRA$ was first tested on the larger, real-life data that were used in the previous chapter, namely the TSP-library data sets $rat575$, $rat783$, $pr1002$ and $rl1323$. For consistency, the $ERRA$ was also tested on our newly generated data sets (first introduced in the previous chapter) to demonstrate how efficient the algorithm is with differently distributed data. Each data set was given a time limit of 24 hours for each value of p, and the result found is either the optimal solution or the best lower bound established in this time limit.

Scenario One: Results using the TSP-Library data sets

Tables 5.8-5.11 are structured in the following way. The first column, titled p, shows the number of required facilities. The second column, Z_H, gives the upper bound found for the best heuristic solution (see Elshaikh *et al* (2016)) that used when generating all the potential solution circles to obtain the next lower bound. The third column, titled Z^*, gives the optimal or best found solution. The final three columns state the total number of iterations of the algorithm to find the corresponding Z^* value, the

total number of times the lower bound was updated and the total computational time for the $ERRA$ respectively.

	Best Heuristic	Enhanced Reverse Relaxation			
p	Z_H	Z^*	# Iterations	# LB Updates	CPU Time (secs)
10	67.926	67.926	110	68	11.42
20	45.621	45.475	342	233	590.42
30	35.556	35.556	335	188	1412.07
40	30.265	30.063	434	253	76647.60
50	26.173	25.826	332	159	16354.00
60	23.662	23.163	294	117	22277.80
70	21.059	20.858	283	100	28210.70
80	19.510	19.026	253	68	9789.67
90	17.923	17.460	239	60	1114.37
100	16.511	16.420	243	53	3696.07
Average	**30.421**	**30.178**	**287**	**130**	**16010.41**

Table 5.8: Results for TSP-Lib $rat575$ using the Enhanced Reverse Relaxation Algorithm

Results show that optimal solutions were found in a reasonable computational time for the larger data sets. The $ERRA$ showed its superiority by discovering optimal results for the first time, for example when $p \leq 30$ for the data set $rl1323$.

	Best Heuristic	Enhanced Reverse Relaxation			
p	Z_H	Z^*	# Iterations	# LB Updates	CPU Time (secs)
10	79.313	79.313	151	102	24.14
20	53.441	53.332	386	268	1181.29
30	42.395	42.307	638	460	39558.80
40	35.962	$35.249^{\perp}$	417	259	86400.00
50	31.184	$30.647^{\perp}$	401	236	86400.00
60	28.053	$27.067^{\perp}$	276	134	86400.00
70	25.446	$24.521^{\perp}$	250	114	86400.00
80	23.560	$22.519^{\perp}$	261	112	86400.00
90	21.710	$20.940^{\perp}$	265	103	86400.00
100	20.334	$19.526^{\perp}$	254	90	86400.00
Average	**36.140**	$---$	**330**	**188**	**64556.42**

$^{\perp}$ Best lower bound found within 86400 seconds.

Table 5.9: Results for TSP-Lib $rat783$ using the Enhanced Reverse Relaxation Algorithm

Furthermore, it can be observed that the fastest computational times to optimally solve the TSP-Library data set was found when $p = 10$ for each instance. These two observations suggest that the enhanced reverse relaxation algorithm is particularly effective for smaller values of p. This is a promising result, especially as the Drezner enhanced algorithm (DEA) developed in Chapter 4 found optimally solving the $p-$centre problem

for smaller values of p more challenging. Therefore, DEA and the $ERRA$ compliment each other.

	Best Heuristic	Enhanced Reverse Relaxation			
p	Z_H	Z^*	# Iterations	# LB Updates	CPU Time (secs)
10	2389.36	2389.36	119	69	17.97
20	1609.54	1607.53	471	336	5066.19
30	1231.36	1231.36	330	185	2072.17
40	1030.40	1021.41	307	159	1091.56
50	901.455	895.342	333	170	10874.10
60	801.474	795.709	328	137	23724.40
70	727.154	725.431	366	142	10545.50
80	664.798	655.746	269	89	1492.98
90	604.152	604.152	270	67	670.39
100	559.017	555.662	256	51	391.07
Average	**1051.870**	**1048.170**	**305**	**141**	**5594.63**

Table 5.10: Results for TSP-Lib $pr1002$ using the Enhanced Reverse Relaxation Algorithm

	Best Heuristic	Enhanced Reverse Relaxation			
p	Z_H	Z^*	# Iterations	# LB Updates	CPU Time (secs)
10	2897.49	2987.49	328	242	666.71
20	1886.82	1868.92	674	543	5485.95
30	1466.97	1466.97	858	670	81009.20
40	1236.38	$1225.74^{\perp}$	666	486	86400.00
50	1060.82	$1051.82^{\perp}$	557	384	86400.00
60	941.870	$930.977^{\perp}$	472	297	86400.00
70	844.967	$841.578^{\perp}$	571	323	86400.00
80	774.764	$770.532^{\perp}$	495	267	86400.00
90	720.625	706.145	428	202	9863.38
100	662.936	$658.267^{\perp}$	424	195	86400.00
Average	**1249.364**	$---$	**547**	**361**	**61542.52**

$^{\perp}$ Best lower bound found within 86400 seconds.

Table 5.11: Results for TSP-Lib $rl1323$ using the Enhanced Reverse Relaxation Algorithm

Further comparison of the results found using $ERRA$ and DEA suggest that both algorithms are similar in efficiency, with the DEA requiring slightly less computational time. However, it is important to note that the total computational time required for the DEA relies on the initial upper bound found using a powerful heuristic, whereas $ERRA$ does not rely on external methods.

Finally, after comparing the best upper bound solution given (Z_H) to the best lower bound found using $ERRA$ within the time limit of 86400 seconds, we can observe that

the lower bound value is always proportionally very close to the value of the best upper bound, with the worst-case being a 4.4% difference ($n = 783, p = 80$). This therefore suggests that the solution found for the instances where the total amount of computational time has been used could be either be optimal or near optimal. It is also worth mentioning that such lower bounds can be used to access the performance of every metaheuristic that will be developed.

Scenario Two: Results using our new generated data sets

Tables 5.12-5.14 display the results for the generated data sets, and are structured in the following way. The first column displays the data's distribution type, which are discussed in Section 4.9.2, Chapter 4. The second column displays the number of facilities located, p. The third column shows the initial upper bound value, Z_1 found using the H_2 heuristic used in the previous chapter, and the fourth column displays corresponding optimal solution, Z^*. This is followed by the number of iterations, lower bound updates and total computational time taken to find that optimal solution respectively using the enhanced reverse relaxation algorithm.

For consistency with Chapter 4, we compare the average computational times spent for each value of n for the three distribution types using Figure 5.17.

		H_2 Heuristic	Enhanced Reverse Relaxation			
Dist	p	Z_1	Z^*	# Iterations	# LB Updates	CPU Time (secs)
Cl	25	56.871	53.677	60	10	1.623
Cl	50	38.082	32.660	94	18	4.929
Cl	75	31.827	24.279	117	15	8.939
Cl	100	25.480	19.963	124	14	8.814
Average	**63**	**38.065**	**32.645**	**98.75**	**14**	**6.08**
S-Cl	25	103.393	89.098	82	23	3.836
S-Cl	50	72.890	53.891	89	16	4.13
S-Cl	75	60.891	38.852	99	9	5.353
S-Cl	100	49.498	29.568	105	8	6.811
Average	**63**	**93.694**	**52.852**	**94**	**14**	**5.03**
R	25	119.474	107.353	190	91	81.508
R	50	82.619	67.151	156	46	28.514
R	75	63.389	51.574	146	26	16.527
R	100	54.722	40.140	127	12	10.069
Average	**63**	**80.051**	**66.554**	**155**	**44**	**34.15**

Table 5.12: Solutions for the generated data set where $n = 400$ using the Enhanced Reverse Relaxation Algorithm

		H_2 Heuristic	Enhanced Reverse Relaxation			
Dist	p	Z_1	Z^*	# Iterations	# LB Updates	CPU Time (secs)
Cl	25	64.070	57.618	106	40	15.48
Cl	50	44.003	36.739	147	41	37.93
Cl	75	34.436	25.986	146	34	28.76
Cl	100	30.566	20.826	162	36	37.51
Average	**63**	**43.269**	**35.292**	**140**	**38**	**29.92**
S-Cl	25	73.135	64.368	198	101	88.62
S-Cl	50	51.587	41.904	210	83	128.36
S-Cl	75	41.135	31.275	168	37	45.16
S-Cl	100	35.203	24.323	139	25	22.98
Average	**63**	**50.273**	**40.468**	**179**	**62**	**71.28**
R	25	74.402	67.979	340	219	654.73
R	50	52.366	44.411	313	146	5570.63
R	75	41.001	33.616	233	73	159.93
R	100	35.249	27.023	191	34	56.89
Average	**63**	**50.005**	**43.257**	**269**	**118**	**1610.54**

Table 5.13: Solutions for the generated data set where $n = 600$ using the Enhanced Reverse Relaxation Algorithm

		H_2 Heuristic	Enhanced Reverse Relaxation			
Dist	p	Z_1	Z^*	# Iterations	# LB Updates	CPU Time (secs)
Cl	25	63.776	60.879	71	6	6.458
Cl	50	48.089	40.265	194	71	80.684
Cl	75	36.634	30.446	236	83	161.411
Cl	100	31.544	24.824	246	63	169.914
Average	**63**	**45.011**	**39.104**	**187**	**56**	**104.62**
S-Cl	25	90.593	81.238	294	167	556.611
S-Cl	50	62.332	51.051	228	88	287.611
S-Cl	75	50.487	38.596	204	63	95.348
S-Cl	100	45.900	31.205	215	63	122.563
Average	**63**	**62.328**	**50.523**	**235**	**95**	**265.53**
R	25	126.385	115.444	370	254	1252.63
R	50	89.059	75.935	407	222	63067.20
R	75	71.218	58.217	335	145	28511.00
R	100	61.890	48.545	315	103	6060.80
Average	**63**	**87.128**	**74.535**	**357**	**181**	**24722.90**

Table 5.14: Solutions for the generated data set where $n = 800$ using the Enhanced Reverse Relaxation Algorithm

Results suggest that the more clustered a data's distribution is, then the quicker the algorithm will find the crucial demand points needed to define the locations of the p facilities and therefore the optimal solution (we note that this is similar to the results for the generated data found using the DEA proposed in Chapter 4, except these findings remain consistent regardless of data set size). This is to be expected, as the demand points that are required to find the optimal solution in a clustered distribution will most likely be located on the outside of clustered areas. Thus their positioning

makes it easier for the enhancement AddE2 to find and add these points to the subset. This is not the case for a randomly distributed data set, and so these distribution types are more challenging to solve optimally.

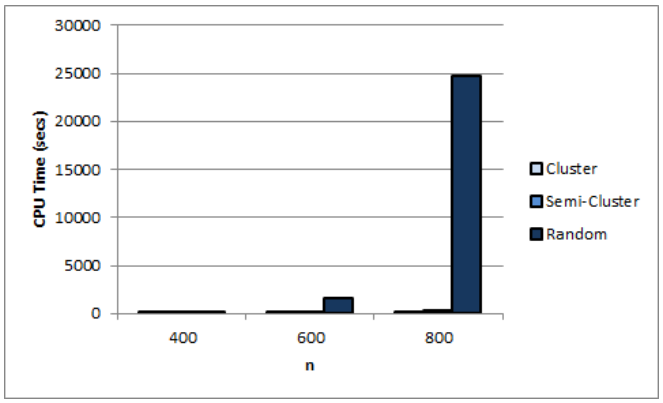


Figure 5.17: Average computational time for generated clustered, semi-clustered and randomly distributed data sets

5.5 Summary

This chapter has investigated the method of relaxation to optimally solve the continuous $p-$centre problem. Recent and effective algorithms that use this method were introduced, and a further investigative work highlighted the best algorithm to enhance with an aim to develop a faster algorithm that could optimally solve large data sets. Four enhancements were introduced, mathematically supported where applicable and tested to show the enhanced reverse relaxation algorithm required less computational time than the original reverse relaxation algorithm suggested by Chen & Chen (2009). The enhanced algorithm was tested on four large real-life data sets from the TSP-Library, namely $pr439$, $rat575$, $rat783$, $pr1002$ and $rl1323$, and six generated data sets to test the efficiency of the algorithm for different distributions. The enhanced algorithm showed its strength by finding optimal solutions for the first time in some of the real-life instances, for example when $p < 30$ for the data set $rl1323$. Investigative work using the generated data sets revealed that the enhanced reverse relaxation algorithm requires less computational time when solving data sets with clustered distributions.

The next chapter will create a new matheuristic, designed to find a good feasible solution that can be embedded into the enhanced algorithms given in Chapters 4 and 5 to obtain the optimal solution.

Chapter 6

A Facility-Based Relaxation Algorithm

6.1 Introduction

This chapter investigates the use of relaxation on the number of facilities, rather than the number of demand points, to solve the $p-$centre problem using a new matheuristic. The matheuristic begins by finding an efficient subset of facilities, and the next section examines the two selection methods used in this study to find this subset. The first option involves a known geometric-based method using the Voronoi diagram and the second option is a simplified version of the first. We then discuss the initial results and observations that led to introducing an adapted matheuristic inspired by the Variable Neighbourhood Search metaheuristic. A diversification method is also developed and incorporated into the matheuristic, and results using the adapted matheuristic are recorded and discussed. Finally, the last section displays the optimal results found for larger data sets from the TSP-Library and the generated data sets, first introduced in Chapter 4, when the feasible solutions generated from the new matheuristic are embedded into the exact algorithms developed in Chapters 4 and 5.

6.2 A New Matheuristic

6.2.1 Overview

For clarity, in this chapter we shall refer to the subset of demand points as Sub and the subset of facilities as $FSub$. Furthermore, for consistency with previous chapters, we define the solution circle of the $p-$centre problem to be the largest covering circle found in the feasible solution, with facility f_{rMax} located at its centre. Let us now define the full set of circles that form the feasible solution to the $p-$centre problem as the set of *solution circles*. Lastly, let us define D_{j^1,j^2} as the Euclidean distance between facility j^1 and facility j^2.

As previously noticed by Drezner (1984a), if a solution to the $p-$centre problem is not optimal, then the size of the largest covering circle (i.e. the solution circle) must be decreased to allow the solution to converge towards the optimal solution. It is this logic that inspired the exploration and development into the new matheuristic, which we shall refer to as the relaxed p' matheuristic (where $p' < p$).

In brief, the matheuristic finds a feasible solution to the $p-$centre problem and identifies the facility f_{rMax}. We obtain $FSub$ from the remaining facilities based on their distance and location to f_{rMax}. The demand points that are covered by this subset of facilities are then treated as a relaxed problem to solve in order to improve the overall solution value. Therefore, this matheuristic targets and improves a specific area defined by the neighbouring space of the solution circle in the current feasible solution. This scheme allows a better feasible solution to be found quickly and efficiently by guiding the solution value as the matheuristic progresses, and therefore allowing it to converge towards the optimal solution.

Figure 6.1 demonstrates the basic idea with an example of the first three iterations of the matheuristic. In iteration 1, the solution circle is shown as the highlighted circle. The set of circles that lie inside the dashed circle form the subset of facilities $FSub$, and the demand points (not shown) inside these circles form the relaxed problem that will be solved optimally. Iteration 2 shows this area is now covered more efficiently, and so there is now a new solution circle, which is highlighted. Again, the dashed circle shows the new subset of facilities, $FSub$, and the demand points covered by $FSub$ are treated as a relaxed problem to be solved optimally. Iteration 3 shows that, once again, there is a new solution circle to continue the process with.

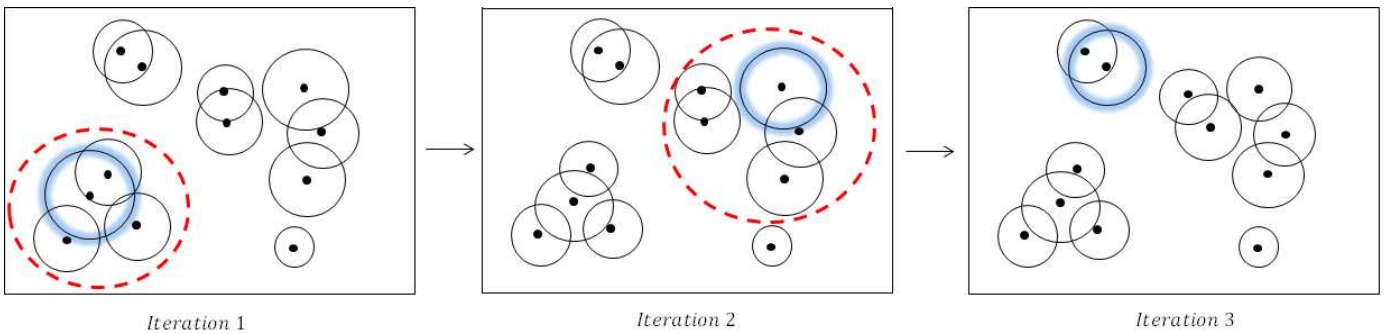
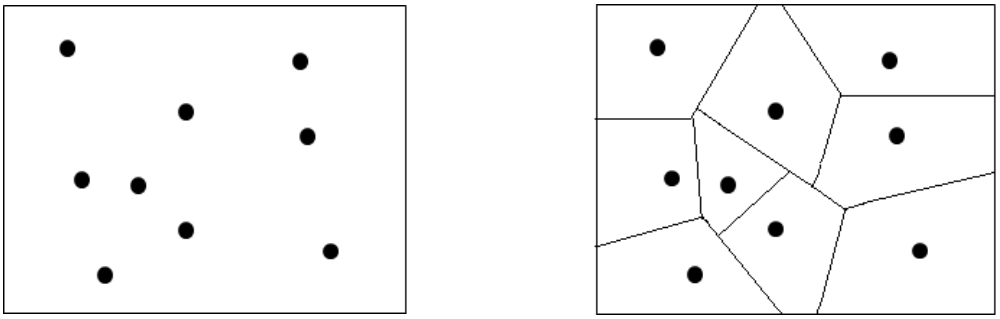


Figure 6.1: The new matheuristic targets specific facilities in the feasible solution and optimally solves the sub-problem

The main challenge of this matheuristic is to find a good subset of facilities ($FSub$) such that an improvement amongst the corresponding subset of demand points would lead to an overall improvement for the full problem. In this study, two methods are adopted to find a suitable subset of facilities. The first one uses the Voronoi diagram, whereas the second one relies on an approximation to define $FSub$. A brief introduction to the Voronoi diagram is outlined in the next sub-section.

6.2.2 The Voronoi Diagram-Based Method

This sub-section will begin with a brief background on the Voronoi diagram. Let there be p facilities in a plane. The **Voronoi diagram** partitions the plane into p polygons with exactly one facility inside each polygon. Each polygon, or *Voronoi polygon*, is defined by the area in the plane that lies closest to its associated facility in terms of Euclidean distance. As an example, Figure 6.2a shows a set of facilities in the plane and its corresponding Voronoi diagram is shown in Figure 6.2b.



(a) (b)

Figure 6.2: A Voronoi diagram example

Let there be a plane of n demand points and p facilities. Each facility is situated inside strictly one polygon, which we refer to as its corresponding Voronoi polygon $Vor_{j'}$ where $j' = 1, \ldots, p$. We can record the demand points $i \in I$ that lie in $Vor_{j'}$ as follows

$$Vor_{j'} = \{i \in I \mid d_{i,j'} < d_{i,k} \text{ for } k = 1, \ldots, p : k \neq j'\}. \tag{6.1}$$

If we connect the centres of the adjacent Voronoi polygons with a straight line, we get the Delaunay graph (or Delaunay triangulation), see Figure 6.3. Two points connected by an edge of the Delaunay graph can be considered "neighbours", inasmuch as their Voronoi polygon's share an edge.

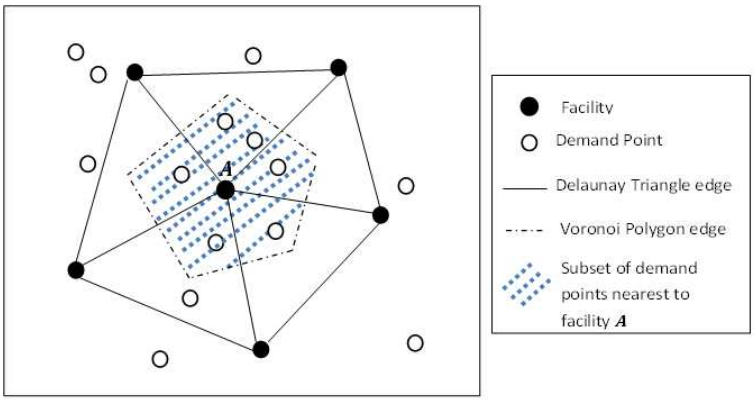


Figure 6.3: The relationship between Delaunay Triangulation and the Voronoi Polygon

There are a number of efficient methods to calculate the Voronoi diagram running in O($n$log$n$) time, such as Fortune's algorithm. Further information on Fortune's algorithm and Voronoi diagrams can be found in Chapter 7 in De Berg *et al* (2000).

The first selection method for the matheuristic adds the facilities to $FSub$ that are associated with the edges of the Voronoi polygon surrounding the facility with the largest circle (i.e. the Delaunay neighbours), and will be referred to as the **Voronoi-based method**. As this requires only a section of the Voronoi diagram to be calculated, Fortune's algorithm was not used as it could not be easily adapted to allow only one specific Voronoi polygon to be found as opposed to computing the full diagram. Instead, we used the following method to find the single required Voronoi polygon.

Once an initial solution for the $p-$centre problem is found, the Voronoi-based method begins by identifying f_{rMax}. The $(p-1)$ midpoints between f_{rMax} and the remaining facilities are then found. For each midpoint, the line of equation that both passes through it and is perpendicular to the line that passes through the given midpoint and f_{rMax} is then calculated. This creates $(p-1)$ half planes. The common intersection between each line is then computed, and the smallest polygon defined by a subset of the intersections that encompasses f_{rMax} is the Voronoi polygon of interest. The intersection points create the Voronoi polygon's corners, and the lines of equation associated with these intersections form the Voronoi polygon's edges. The facilities associated with the Voronoi polygon edges are added to $FSub$. The detailed procedure of the Voronoi-based method is given in Figure 6.4.

1. Input: F. Set $F' = F$. Define the facility centre points as $(x^c_{f'}, y^c_{f'})$ where $f' \in F'$.
2. Identify the facility, f_{rMax}, associated with the solution circle. Set $FSub = \{f_{rMax}\}$ and $F' = F' \setminus \{f_{rMax}\}$.
3. For each $f' \neq f_{rMax}$, find the set of midpoints $(X_M \times Y_M)$ as follows

$$X_M = \{x^M_{f'} : x^M_{f'} = \frac{x^c_{f'} + x^c_{f_{rMax}}}{2}\},$$

and

$$Y_M = \{y^M_{f'} : y^M_{f'} = \frac{y^c_{f'} + y^c_{f_{rMax}}}{2}\}.$$

4. Find the set of gradients for the lines that pass through each midpoint and f_{rMax}.

$$G = \{m_{f'} : m_{f'} = \frac{y^M_{f'} - y^c_{f_{rMax}}}{x^M_{f'} - x^c_{f_{rMax}}} \ \forall \ f' \neq f_{rMax}\}.$$

5. Find the set of gradients for the perpendicular lines in G, namely $G^\perp$ where

$$G^\perp = \{m^\perp_{f'} : m^\perp_{f'} = -\frac{1}{m_{f'}} \ \forall \ m_{f'} \in G\}.$$

6. Find all intersection points of lines determined by point $(x^M_{f'}, y^M_{f'})$ and gradient $m^\perp_{f'} \ \forall \ f' \in F'$.
7. While $|F'| \neq 0$ do:
   i) Find the closest intersection point to (x^c_{rMax}, y^c_{rMax}). Let the two lines that form this intersection be l_1 and l_2, and their respective facilities f_1 and f_2. Set $FSub = FSub \cup \{f_1, f_2\}$ and $F' = F' \setminus \{f_1, f_2\}$.
   ii) Identify which side of the line f_{rMax} lies on for l_1. For all $f' \in F'$ that sit on the opposite side of l_1, set $F' = F' \setminus \{f'\}$. Repeat for line l_2.
8. Return $FSub$.

Figure 6.4: The Voronoi-based Method

Figure 6.5 demonstrates finding $FSub$ based on facility f_{rMax} using the Voronoi-based method. The perpendicular lines that pass through the midpoints of f_{rMax} and the required facilities are shown as hashed lines. The intersection of these hashed lines highlights the Voronoi polygon area that encompasses f_{rMax}. The facilities that are added to $FSub$ in this example are represented by highlighted circles. We can see in this instance that $|FSub| = 6$ when selecting facilities using the Voronoi-based method.

The time complexity of the Voronoi-based method is bounded by Step 6 in Figure 6.4, and so has a time complexity of $O(p^2)$.

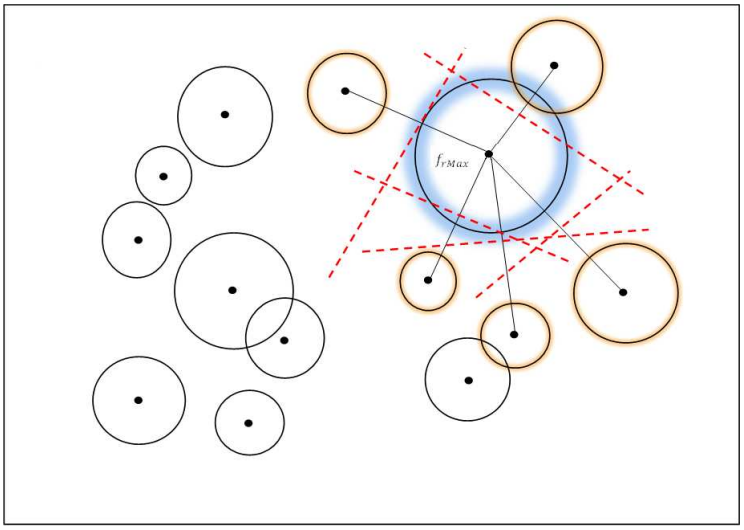


Figure 6.5: Finding $FSub$ using the Voronoi-based method

6.2.3 The Simpler Neighbouring Facilities Method

The second method, which is a simpler and approximate version to the Voronoi-based method, is referred to as the **simpler neighbouring facilities method**. Facilities are chosen based on their distance and location to f_{rMax} in a less complicated way, thus making it easier to identify the facilities for $FSub$. The process is described in Figure 6.6.

1. Input: F. Set $F' = F$.
2. Identify the facility, f_{rMax}, associated with the solution circle. Set $FSub = \{f_{rMax}\}$ and $F' = F' \setminus \{f_{rMax}\}$.
3. While $|F'| \neq 0$ do:
   i) Find $f^* = Arg((\underset{f' \in F'}{Min}\{D_{f',f_{rMax}}\}))$ and set $FSub = FSub \cup \{f^*\}$ and $F' = F' \setminus \{f^*\}$.
   ii) Find the line that passes through f_{rMax} and f^*, say l.
   iii) Find the perpendicular line to l that passes through f^*, say $l^{\perp}$.
   iv) For all $f' \in F'$ lying on the opposite side of $l^{\perp}$ to f_{rMax}, set $F' = F' \setminus \{f'\}$.
4. Return $FSub$.

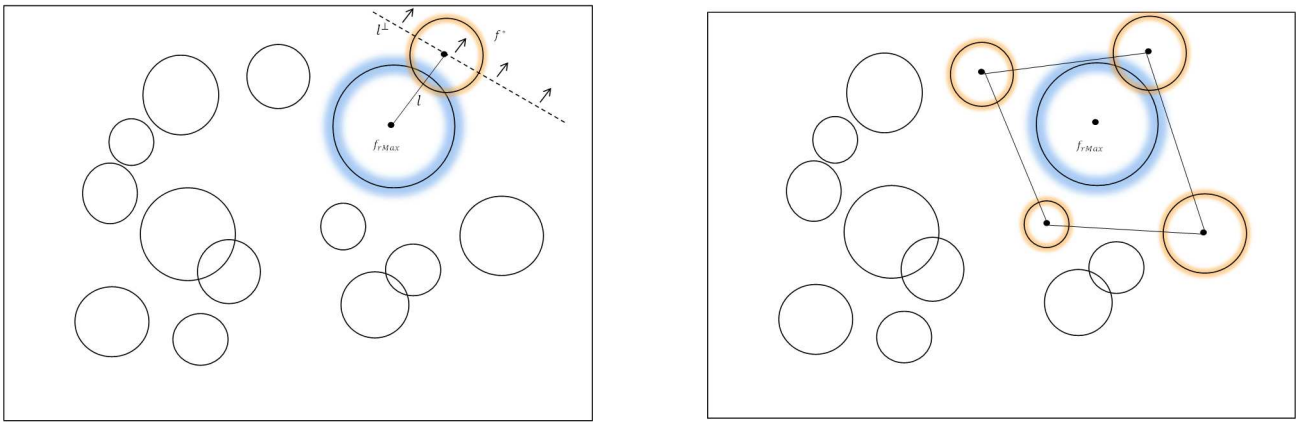
Figure 6.6: The Simpler Neighbouring Facilities Method

Much like the Voronoi-based method, the simpler neighbouring facilities method begins

by identifying f_{Max}. From the set of potential neighbouring facilities, F', the algorithm then finds the closest facility, f^*, to f_{Max}. The line, l, that passes through f_{Max} and f^* is constructed, allowing us to obtain the line, $l^{\perp}$, that both passes through f^* and is perpendicular to line l. We then determine which side of $l^{\perp}$ the facility f_{Max} lies on, and all $f' \in F'$ that lie on the opposite side to f_{Max} are deleted from F' as potential neighbouring facilities. The process continues until $F' = \emptyset$.

For clarity, the new method of establishing the simpler neighbouring facilities, or neighbouring facilities, of f_{rMax} is illustrated in Figures 6.7a & 6.7b using the same configuration as in Figure 6.5. In Figure 6.7a, the closest facility to f_{rMax} is determined and labelled as f^*. The line, l, that passes through both facilities is found, as is the perpendicular line, $l^{\perp}$, that passes through facility f^*. As f_{rMax} lies below $l^{\perp}$, any facilities that lie above (as the arrows indicate) must be eliminated from the search as they cannot be a neighbouring facility.

Figure 6.7b shows all the neighbouring facilities of f_{rMax}. The intersection of the lines of the equation passing through their centres show the corners defining the neighbouring facilities polygon. In this example, we can see $|FSub| = 5$ using the neighbouring facilities selection method. In other words, one more facility is identified using the Voronoi-based method.



(a) Finding One Neighbouring Facility for f_{rMax} (b) All Neighbouring Facilities for f_{rMax}

Figure 6.7: Finding $FSub$ using the Neighbouring Facilities Method

Note that the time complexity of the simpler neighbouring facilities method is bounded by calculating the closest facility to f_{Max} and so has a complexity of $O(p)$. Further-

more, note that other neighbourhood graphs exist in the literature, such as the Gabriel Graph proposed by Gabriel & Sokal (1969), and the Relative Neighbourhood Graph proposed by Toussaint (1980).

These two facility selection methods shall be compared for their strengths and weaknesses in the next section, but first we shall describe the new matheuristic that incorporates such a partitioning.

6.2.4 The Relaxed p' Matheuristic

The algorithm for the new matheuristic, which we refer to as **the relaxed p' matheuristic**, is described in Figure 6.8.

1. Find a feasible solution yielding the initial set of facilities F where $|F| = p$.
2. Find $FSub$ using the Voronoi-based method (Figure 6.4), or the simpler neighbouring facilities method (Figure 6.6) where $p' = |FSub|$.
3. Set $Sub = \{i \in I : d_{i,f'} \leq r_{f'} \forall\ f'\ \in FSub\}$.
4. Solve the $p'-$centre problem for Sub. This yields new p' facility locations Δ where $\Delta = \{\delta_1, \ldots, \delta_{p'}\}$ and solution value Z.
5. Set $F = (F \setminus FSub) \cup \Delta$.
6. Locate-allocate procedure: Allocate all $i \in I$ to their closest $f \in F$ and find a new feasible solution and set of facilities F. Repeat until the configuration of the solution circles do not change.
7. Check stopping criteria.
    (a) If there is no change in the facility configuration, return the best feasible solution found and stop.
    (b) Else, record the solution value and configuration, and go back to Step 2.

Figure 6.8: The Relaxed p' Matheuristic

The proposed matheuristic first finds an initial solution which can be achieved using a simple heuristic or a more powerful metaheuristic if necessary. The facility f_{rMax} is identified, and $FSub$ is established using either the Voronoi or the simpler neighbouring facilities method described in the previous sub-sections. As stated previously, the

demand points that are encompassed by the facilities in $FSub$ form a subset of demand points which are optimally solved using the classic $p-$centre formulation, For_{pc} (see Section 3.2.2, Chapter 3) where, in this instance, $p = |FSub|$. This yields an improved solution value for $FSub$ (i.e. the radius of the largest covering circle is reduced) and creates new positions for the facilities in the subset F. These new positions replace the original positions, and a locate-allocate heuristic is then used to identify the new and improved feasible solution for the full problem. This process is repeated until there is no further improvement is found (i.e. the configuration of the solution circles do not change).

In this study, an initial solution is found using the H_2 heuristic to remain consistent with results from previous chapters. This is so a comparison between the different methods developed in this thesis can be as fair and as accurate as possible.

A simple improvement for the relaxed p' matheuristic

Before any initial results were found using the relaxed p' matheuristic, an obvious improvement was incorporated into the matheuristic. Instead of solving the subset optimally using the traditional formulation (For_{pc}), this was solved faster using the Enhanced Reverse Relaxation Algorithm ($ERRA$) given in Figure 5.16, Chapter 5. Therefore, Step 4 in Figure 6.8 was changed to Step 4', as this allows the subset to be optimally solved in less computational time and thus creates a more efficient algorithm.

> Step 4'. Solve the $p'-$centre problem for Sub using the $ERRA$ described in Figure 5.16 in Chapter 5. This yields new p' facility locations Δ where $\Delta = \{\delta_1, \ldots, \delta_{p'}\}$ and solution value Z.

It is important to note that the overall solution of the relaxed p' matheuristic is affected by the initial solution's largest facility and its surrounding facilities.

A basic adjustment to the ERRA

A small adjustment was made to the $ERRA$ with respect to the minimum number of demand points to be added to the subset. Here, a minimum of one, rather than two, points is used as instances occur where there is only one demand point not covered by the current solution, and so only one point must be added to the subset. These

instances are more likely to occur if the cardinality of Sub is small.

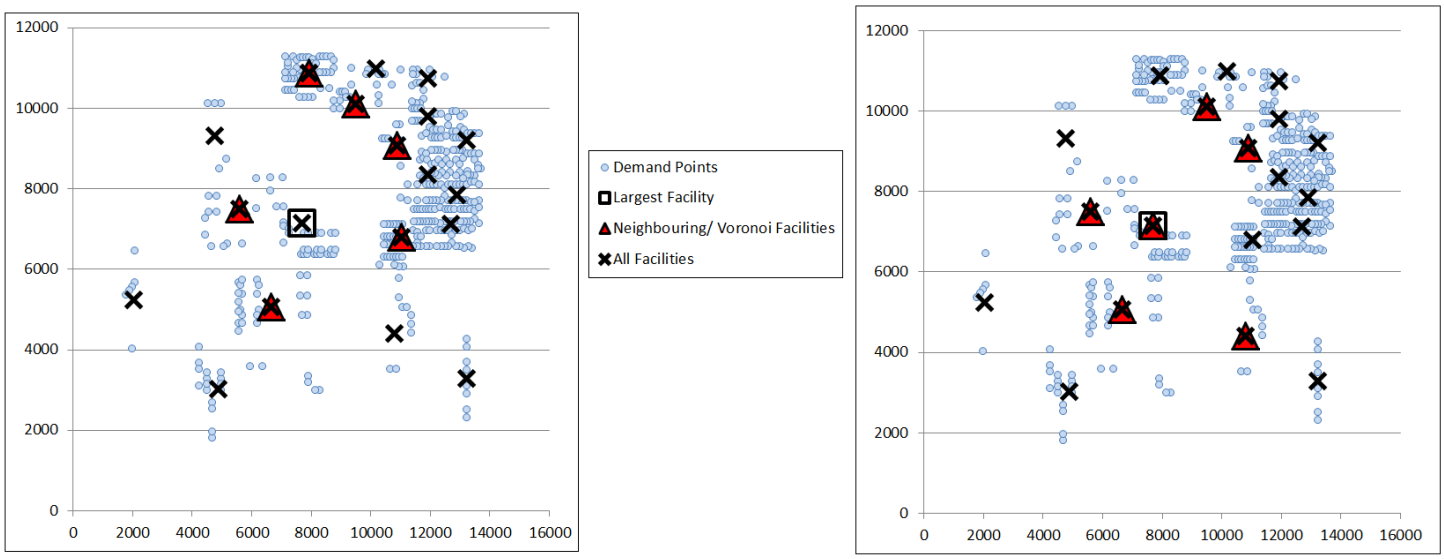
6.3 Initial Implementation & Observations

The relaxed p' matheuristic was first implemented for the TSP-Library data set $pr439$ where $p = 10, 20, \ldots, 100$, and the results are given in Table 6.1. An initial solution was found using 100 iterations of the H_2 heuristic, and this solution value, Z_1, is displayed in the second column of Table 6.1. A comparison of the results found with the relaxed p' matheuristic using the simpler neighbouring facilities method and the Voronoi-based method are given in columns $3-6$ and columns $7-10$ respectively.

	H_2	Simpler Neighbouring Facilities Method				Voronoi Method			
p	Z_1	$Z^{1^{st}}$	$\lvert FSub_1\rvert$	$Z^{\perp}$	# Iterations	$Z^{1^{st}}$	$\lvert FSub_1\rvert$	$Z^{\perp}$	# Iterations
10	1990.290	1990.290	4	1990.290	2	1990.290	3	1990.290	2
20	1453.530	1453.530	5	1453.530	2	1453.530	3	1453.530	2
30	1226.020	1226.020	4	1226.020	2	1226.020	3	1226.020	2
40	1211.960	1126.730	4	1008.170	5	1126.730	4	1125.280	4
50	1125.280	1008.170	5	1008.170	3	1008.170	5	1008.170	3
60	760.345	760.345	4	760.345	2	760.345	7	760.345	2
70	608.790	584.433	6	584.433	3	584.433	5	584.433	3
80	606.347	606.347	3	606.347	2	606.347	3	606.347	2
90	570.008	570.008	3	570.008	2	570.008	5	570.008	2
100	570.008	570.008	3	570.008	2	570.008	5	570.008	2
Average	**1012.258**	**992.024**	**4**	**977.732**	**2**	**989.588**	**4**	**989.443**	**2**

Table 6.1: First results using the relaxed p' matheuristic for $pr439$

Columns 3 & 7 and 4 & 8 in Table 6.1 show the first upper bound found, $Z^{1^{st}}$, and the size of the first subset of facilities, $FSub_1$, found with the simpler neighbouring facilities method and the Voronoi-based method respectively. This gives a fair and interesting comparison as both facility selection methods started with the same initial solution. Columns 5 & 9 show the solution value found, $Z^{\perp}$, using the relaxed p' matheuristic. The total number of iterations of the matheuristic is given in columns 6 & 10 for the simpler neighbouring facilities method and the Voronoi-based method respectively.



(a) The Neighbouring Facilities Method ($|FSub| = 7$)

b) The Voronoi Method ($|FSub| = 6$)

Figure 6.9: An example when the neighbouring facilities method yields a larger $FSub$

Two main observations can be concluded from the information given in Table 6.1. Firstly, we observe that sometimes $|FSub|$ is larger when using the simpler neighbouring facilities method rather than the Voronoi-based method. This is due to the positioning of the facilities around f_{rMax}. An example can be seen in Figure 6.9, where Figure 6.9a shows the facilities found using the simpler neighbouring facilities method and Figure 6.9b shows the facilities found using the Voronoi-based method. Due to both the positioning of f_{rMax} and the distribution of the data, the simpler neighbouring facility method picks a larger, and less efficient, subset to the Voronoi-based method. This small example highlights the potential strengths the Voronoi-based method may have in comparison to the simpler neighbouring facilities method.

Our second observation from Table 6.1 discovers that the matheuristic cannot run for many iterations as it becomes caught in a local minimum very quickly (i.e. the configuration of the full problem does not change between the locate-allocate heuristic solution and when solving the subset optimally). As this greatly affects the overall capability of the matheuristic, further refinements need to be looked at in order to create a more efficient algorithm that does not become stuck easily in a local minimum. One way to leave a local minimum is to change the neighbourhood of the subset of facilities. Several ways in which the neighbourhood can be changed and enlarged in order to escape a local minimum are investigated in the next section.

6.4 Changing the Neighbourhood of $FSub$

If the matheuristic gets stuck in a local minimum, additional facilities from the neighbouring area can be introduced to $FSub$ using several different approaches. As this is vital to keep the matheuristic running, we investigated three different variants which will be explained in detail and individually tested. All three options were inspired by the Variable Neighbourhood Search (VNS) metaheuristic.

When the VNS becomes stuck at a local minimum, it systematically uses a larger neighbourhood in order to escape. In other words, a specified number of points (e.g. 1 point) around the current solution is generated. This forms the first neighbourhood of the initial solution, N_1. If an improved solution cannot be found with N_1, then the problem is enlarged by randomly generating a second specified number of points (e.g. 2 points) to create the second neighbourhood, N_2. The problem continues to generate larger neighbourhoods until either a better solution is found or stopping criterion is reached. Further information on the Variable Neighbourhood Search can be found in Salhi (2006) and references therein, and various variations and their applications in Hansen *et al* (2010).

6.4.1 A Randomly Generated Neighbourhood (Variant (a))

The first variant for our matheuristic finds the neighbourhood $N^R_{p^\perp}(x)$ where $p^\perp$ new facilities are randomly added to $FSub$ where $p^\perp = 1, 2, \ldots, p^{max}$. In other words, if the matheuristic is stuck in a local minimum, one facility is added at random to $FSub$. If the heuristic continues to be stuck in a local minimum, two facilities are added at random to $FSub$ instead. This process continues until either the algorithm escapes the local minimum or the maximum number of facilities (p^{max}) has been added to $FSub$. The process of this approach, which we refer to as for variant (a), is described in Figure 6.10. The technique used to determine p^{max} is given next.

Determining the value of threshold p^{max}

As the matheuristic is a relaxation-based approach, we wish to improve the feasible solution value whilst minimising the cardinality of Sub as much as possible. However, as the matheuristic progresses, the feasible solutions tend to be closer to the optimal

1. Input: $FSub$, F. Set $p^{\perp}$ =1.
2. Randomly select $f \notin FSub$ and set $FSub = FSub \cup \{f\}$. Repeat $p^{\perp} - 1$ times.
3. Complete Steps $(3-6)$ in Figure 6.15.
   If there is no change in the configuration of the full feasible solution and p^{max} has not been reached, go to Step 4.
   Else, set $FSub = \emptyset$ and go to Step 2 in Figure 6.15.
4. Remove the $p^{\perp}$ facilities from $FSub$ selected in Step 2. Set $p^{\perp} = p^{\perp} + 1$. and go to Step 2.

Figure 6.10: Changing the Neighbourhood Randomly (Variant (a))

solution and so more facilities need to be added to $FSub$ in order to escape a local minimum. In other words, a maximum number of demand points in Sub must be established so that the algorithm halts when the relaxed problem becomes too large to be solved in reasonable time.

Let Sub_t be the subset of demand points at iteration t and let $N' = N - |Sub_1|$ (i.e. the number of remaining demand points not in Sub after the first iteration).
If

$$|Sub_t| > |Sub_1| + \frac{N'}{100} \times 30, \tag{6.2}$$

then the number of demand points in the subset has exceeded the maximum. In other words, Sub is allowed to increase by an extra 30% of the original number of the remaining demand points. Therefore, p^{max} is the number of facilities in $FSub$ that causes the size of Sub to go over this threshold.

This threshold is incorporated into Step 3 of the Enhanced Relaxed p' Matheuristic.

6.4.2 A Deterministically Generated Neighbourhood (Variant (b))

The second variant finds the neighbourhood $N^D_{p^{\perp}}$ where the $p^{\perp}$ facilities that have the closest sum distance to all facilities in $FSub$ are added to the subset of facilities. The use of the sum distance allows the facilities added to $FSub$ to be both close to f_{rMax}

whilst able to be spread apart from one another. This approach therefore finds facilities for $FSub$ in a deterministic way rather than the random approach given in variant (a). The process for variant (b) is given in Figure 6.11.

1. Input: $FSub$, F.
2. Find f such that
$$f = Arg(\underset{f \in F}{Min}(\sum_{f' \in FSub} D_{f,f'} : f \notin FSub)).$$
Set $FSub = FSub \cup \{f\}$.
3. Complete Steps $(3-6)$ in Figure 6.15.
If there is no change in the configuration of the full feasible solution and p^{max} has not been reached, go to Step 2.
Else, set $FSub = \emptyset$ and go to Step 2 in Figure 6.15.

Figure 6.11: Changing the Neighbourhood Deterministically (Variant (b))

It is important to note that another possible option is to find the $p^{\perp}$ closest circles to f_{rMax}. There is, however, a potential weakness for this approach as shown using the following worst-case scenario example.

An Illustrative Example

Figure 6.12 shows the facility with the largest covering cirlce, f_{rMax}, and the set of selected facilities $FSub$ with the highlighted circles. If the closest facility to f_{rMax} is selected, the facility labeled F_1 would be added to $FSub$. As this facility lies close to F', it is likely that the heuristic will remain stuck in the local optimum and therefore require another facility to be added to $FSub$. In this instance, the two facilities with the closest distance to $FSub$ would be facility F_1 (found previously) and F_2. Following that, the three facilities that have the closest distances are F_1, F_2 and F_3. This may continue to cause difficulty escaping the local minimum as the facilities selected form a cluster in a small area of the plane. Therefore, it is far more desirable to use the closest sum distance such that a compromise is found between selecting a facility that lies in a close proximity to f_{rMax} whilst being spread out from the other facilities in $FSub$.

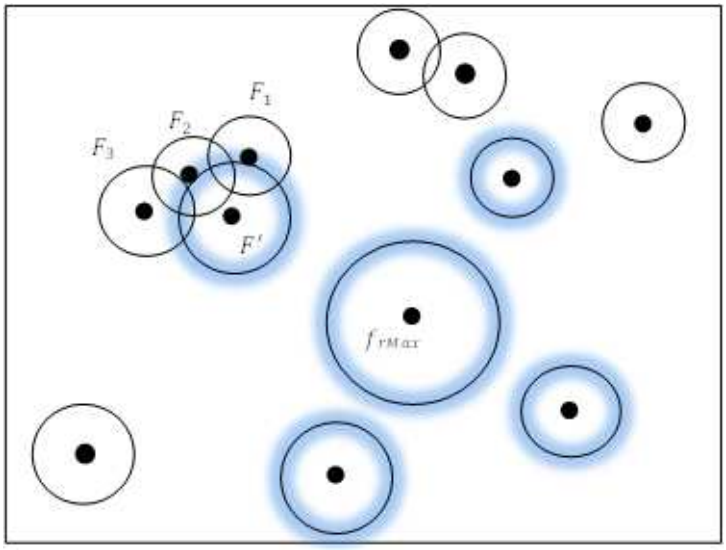


Figure 6.12: Finding the closest to f_{rMax}

6.4.3 Generating a Neighbourhood using Alternating Methods (Variant (c))

The third variant finds a new neighbourhood by alternating between variants (a) and (b). This method therefore wishes to find a balance between locating facilities close to f_{rMax} whilst incorporating randomisation to discourage the matheuristic from becoming caught in a local minimum. Therefore, we define the neighbourhood $N^A_{p^\perp}$ to be

$$N^A_{p^\perp} = \begin{cases} N^R_{p^\perp} \text{ if } p'' \text{mod } 2 = 1, \\ N^D_{p^\perp} \text{ if } p'' \text{mod } 2 = 0 \ , \end{cases} \tag{6.3}$$

where p'' is the number of consecutive times the neighbourhood is enlarged, $k' = p''\text{mod}2$ and $p^\perp = \frac{p''+k'}{2}$. Thus we get the neighbourhood structure $N^A_{p^\perp} = \{N^R_1, N^D_1, N^R_2, N^D_2, \ldots, N^R_{p^{max}}, N^D_{p^{max}}\}$. The process for variant (c) is given in Figure 6.13.

Example

Let the matheuristic be stuck in a local minimum, and let us wish to change the neighbourhood using variant (c). According to Figure 6.13, variant (c) begins with $p'' = 1$. Therefore, we can see that $k' = 1\text{mod}2 = 1$ and $p^\perp = \frac{1+1}{2} = 1$. We therefore change the neighbourhood using N^R_1 (i.e. randomly add one facility to $FSub$).

1. Input: $FSub$, F. Set $p^{\perp}$ =1.

2. If $p^{\perp} = 1$ mod 2 do:

   Randomly select $f \notin FSub$ and set $FSub = FSub \cup \{f\}$. Repeat $p^{\perp} - 1$ times.

   Else,

   Find f such that

$$f = Arg(\underset{f \in F}{Min}(\sum_{f' \in FSub} D_{f,f'} : f \notin FSub)).$$

   Set $FSub = FSub \cup \{f\}$. Repeat $p^{\perp} - 1$ times.

3. Complete Steps $(3-6)$ in Figure 6.15.
   If there is no change in the configuration of the full feasible solution and p^{max} has not been reached, go to Step 4.
   Else, set $FSub = \emptyset$ and go to Step 2 in Figure 6.15.

4. Remove the $p^{\perp}$ facilities added to $FSub$ in Step 2. Set $p^{\perp} = p^{\perp} + 1$. Go to Step 2.

Figure 6.13: Changing the Neighbourhood Alternately (Variant (c))

6.4.4 The Enhanced Relaxed p' Matheuristic & Diversification

As stated previously, the algorithm for the relaxed p' matheuristic stops if $|Sub| > |Sub_1| + \frac{N'}{100} \times 30$. This is because there are too many demand points in Sub and thus the use of the relaxation method is negated as the problem becomes too large to solve in a reasonable amount of time.

1. Input: I, L, Div^{max}, F and set $It = 0$.

2. If $L = Div^{max}$, return best feasible solution and stop.
   Else, set $L = L + 1$.

3. Find $FSub$ using the Voronoi-based method (Figure 6.4), or the neighbouring facilities method (Figure 6.6) where $|FSub| = p'$. Set $F = FSub$.

4. Find $Sub_2 = \{i \in I : d_{i,f'} > r_{f'} \ \forall \ f' \in F\}$.

5. Randomly select $i \in Sub_2$ and set $F = F \cup \{i\}$. Repeat until $|F| = p$.

6. Locate-allocate all $i \in I$ to their closest $f \in F$ to find a new feasible solution and set of facilities F. Go to Step 2 in Figure 6.15.

Figure 6.14: The Diversification Method

However, in an attempt to tighten the solution value further and find a better feasible solution, a diversification method was developed. This means that in the instance where $FSub$ becomes too big, instead of halting the algorithm we can diversify the solution. This therefore allows more opportunity for the algorithm to find an improved feasible solution. In brief, the diversification method randomly selects new locations for the facilities $f \notin FSub$. For clarity, each time the matheuristic is diversified, we shall refer to it as entering a new loop, L, and the total number of times the matheuristic is allowed to diversify is Div^{max}. Therefore, once $L = Div^{max}$, the algorithm returns the best feasible solution found and stops.

The full description for the diversification method is presented in Figure 6.14. The enhanced relaxed p' matheuristic algorithm that incorporates the different neighbourhood selection variants and the diversification method is given in Figure 6.15.

1. Input: N, Div^{max}. Set $FSub = \emptyset$, $It = 0$ and $L = 1$. Find a feasible solution yielding the initial set of facilities F where $|F| = p$.
2. Find $FSub$ using the Voronoi-based method (Figure 6.4), or the neighbouring facilities method (Figure 6.6) where $|FSub| = p'$.
3. Set $It = It + 1$ and $Sub_{It} = \{i \in I : d_{i,f'} \leq r_{f'} \forall\ f' \in FSub\}$. If $It = 1$, record $|Sub_1|$.
   (a) If $|Sub_{It}| > |Sub_1| + (30 \times \frac{N-|Sub_1|}{100}))$, diversify the solution using the diversification method given in Figure 6.14.
   (b) Else, go to Step 4.
4. Solve the $p'-$centre problem for Sub using the $ERRA$. This yields new p' facility locations Δ where $\Delta = \{\delta_1, \ldots, \delta_{p'}\}$ and solution value Z.
5. Set $F = (F \setminus FSub) \cup \Delta$.
6. Locate-allocate procedure: Allocate all $i \in I$ to their closest $f \in F$ and find a new feasible solution and set of facilities F. Repeat until the configuration of the solution circles do not change.
7. If there is a change in the configuration between Steps 5 & 6, set $FSub = \emptyset$ and go back to Step 2.
   Else, change the neighbourhood of $FSub$ using either variant (a) (Figure 6.10), variant (b) (Figure 6.11) or variant (c) (Figure 6.13).

Figure 6.15: The Enhanced Relaxed p' Matheuristic

6.5 The Enhanced Relaxed p' Matheuristic Results

This section will discuss the results found using the enhanced relaxed p' matheuristic. Firstly, we shall test the different variations of generating a new neighbourhood in order the establish the most efficient method. We shall then investigate the robustness of the solution found using the enhanced relaxed p' matheuristic by embedding its solution into exact methods. Finally, we shall conclude with computational experiments on the larger TSP-Library data sets and the generated data sets.

6.5.1 Allowing Changing Neighbourhoods & Incorporating the Diversification Method

The relaxed p' matheuristic was implemented for the data set $pr439$ for $p = 10, 20, \ldots, 100$ where the neighbourhood was changed when the matheuristic became caught in a local minimum using the three variants described in the previous section. The diversification method was also incorporated into the matheuristic where $Div^{max} = 10$.

For a fair comparison, the same initial solutions that were found using H_2 were recycled. Tables 6.2, 6.3 and 6.4 show the solution value found using the relaxed p' matheuristic using both the simpler neighbouring facilities method and the Voronoi-based method where the neighbourhoods were changed using variants (a), (b) and (c) respectively.

	H_2	Simpler Neighbouring Facilities Method				Voronoi Method			
p	Z_1	$Z^{\perp}$	CPU Times (secs)[a]	# Average Total Iterations	Best Div	$Z^{\perp}$	CPU Times (secs)[a]	# Average Total Iterations	Best Div
10	1990.290	1716.510	11.83	78	9	1790.620	7.83	46	3
20	1453.530	1125.280	22.73	113	6	1029.715	29.06	166	5
30	1226.020	818.917	64.74	372	9	760.345	48.58	399	7
40	1211.960	619.770	64.43	525	5	621.742	69.87	564	2
50	1125.280	497.808	85.75	941	8	473.014	120.35	899	7
60	760.345	426.651	106.26	1063	2	426.651	92.88	963	2
70	608.790	362.500	173.59	1639	5	363.361	127.95	1341	8
80	606.347	327.304	197.02	1823	9	347.311	137.24	1396	4
90	570.008	291.815	215.73	2101	6	290.743	210.28	1991	3
100	570.008	268.386	300.63	2755	9	267.783	277.48	2430	9
Average	**1012.258**	**645.494**	**124.27**	**1141**	**7**	**637.128**	**112.15**	**1018**	**5**

[a] This excludes computational time for the H_2 heuristic.

Table 6.2: Results for the Relaxed p' Heuristic for $pr439$ with Changing Neighbourhood (Variant (a) and the Diversification Method ($Div^{max} = 10$))

	H_2	Simpler Neighbouring Facilities Method				Voronoi Method			
p	Z_1	$Z^{\perp}$	CPU Times (secs)[a]	# Average Total Iterations	Best Div	$Z^{\perp}$	CPU Times (secs)[a]	# Average Total Iterations	Best Div
10	1990.290	1761.360	28.11	105	3	1790.620	9.13	80	7
20	1453.530	1226.020	45.83	168	1	1226.020	25.13	156	2
30	1226.020	900.087	42.92	258	2	874.271	52.87	391	4
40	1211.960	606.347	66.99	460	9	621.742	41.57	337	4
50	1125.280	544.576	59.26	580	10	606.347	62.49	583	1
60	760.345	437.500	131.56	1142	7	606.347	56.09	560	8
70	608.790	362.500	192.80	1519	2	362.500	339.85	2419	7
80	606.347	427.466	186.67	1726	4	400.000	105.81	950	6
90	570.008	353.736	128.63	1228	1	314.740	184.54	1572	1
100	570.008	268.386	348.56	1981	4	353.740	137.70	1165	1
Average	**1012.258**	**688.798**	**123.13**	**917**	**4**	**715.633**	**101.52**	**821**	**4**

[a] This excludes computational time for the H_2 heuristic.

Table 6.3: Results for the Relaxed p' Heuristic for $pr439$ with Changing Neighbourhood (Variant (b) and the Diversification Method ($Div^{max} = 10$))

	H_2	Simpler Neighbouring Facilities Method				Voronoi Method			
p	Z_1	$Z^{\perp}$	CPU Times (secs)[a]	# Average Total Iterations	Best Div	$Z^{\perp}$	CPU Times (secs)[a]	# Average Total Iterations	Best Div
10	1990.290	1716.510	38.72	97	5	1716.510	23.97	147	6
20	1453.530	1125.280	23.52	218	3	1088.720	27.49	203	6
30	1226.020	739.193	63.69	522	7	809.514	42.81	468	8
40	1211.960	580.005	83.56	878	8	580.005	120.68	1055	8
50	1125.280	481.047	100.97	1117	2	481.047	124.56	1265	3
60	760.345	427.566	164.68	1798	3	424.632	190.32	1996	10
70	608.790	362.500	284.81	2716	2	362.500	252.60	2533	10
80	606.347	316.258	329.35	3049	8	332.283	309.15	2996	4
90	570.008	282.013	472.65	4200	2	282.013	449.69	3768	6
100	570.008	265.754	542.84	4663	8	265.754	423.84	3587	7
Average	**1012.258**	**629.613**	**210.48**	**1926**	**5**	**634.298**	**196.49**	**1802**	**7**

[a] This excludes computational time for the H_2 heuristic.

Table 6.4: Results for the Relaxed p' Heuristic for $pr439$ with Changing Neighbourhood (Variant (c) and the Diversification Method ($Div^{max} = 10$))

The tables are organised in the following way. The first two columns show the number of facilities located, p, and the second column shows the initial upper bound value found using the H_2 heuristic. The next eight columns display the solution value found using the enhanced matheuristic, $Z^{\perp}$, the total computational time for the matheuristic only (i.e. not including the H_2 heuristic), the total number of iterations and the loop number that obtained the best solution using the simpler neighbouring method and the Voronoi method respectively.

Tables 6.2−6.4 show that variant (b) produced the weakest result as it yielded the largest upper bound values with the least amount of iterations on average. This result

is expected as variant (b) focuses purely on a specific area of the feasible solution and hence never allows a wider spread of facilities to be added to $FSub$. This leads to the matheuristic becoming stuck at a local minimum more easily in comparison to variants (a) or (c).

Variants (a) and (c) both show strengths and weaknesses. Variant (a) produced good solution values for both facility selection methods and took the least computational time on average out of all three variants. However, variant (c) did allow many more iterations on average which increased the overall computational time, but yielded the best solution values. In other words, variant (c) establishes a good compromise between a deterministic and randomised selection for $FSub$, and therefore was chosen as the variant in subsequent iterations.

6.5.2 Integrating the Relaxed p' Matheuristic with the Optimal Methods - Initial Results

As stated previously, the purpose of the new matheuristic is to find a tight feasible solution such that either a) the solution value found can be used as an initial upper bound for the Drezner enhanced algorithm to solve the problem optimally (see Chapter 4) or b) we use the critical points of the solution circles to create an initial subset of demand points for the $ERRA$ to solve the problem optimally (see Chapter 5).

Table 6.4 shows that the simpler neighbouring facilities method found better solution values for the relaxed p' matheuristic compared to the Voronoi-based method. However, as there is a limited investigative work into this comparison it is worth testing both facility selection methods initially.

The result from the relaxed p' matheuristic was incorporated into the Drezner enhanced algorithm (where the initial upper bounds are the $Z^{\perp}$ values for the corresponding facility selection method taken from Table 6.4) and the $ERRA$. Table 6.5 shows the optimal solution for each value of p under the column titled Z^*. For reference, the previous best computational times spent finding the exact solution using the two exact algorithms stated above (taken from Chapters 4 and 5) are given in columns 3

		CPU Time (secs)					
		Drezner Enhanced			Enhanced Reverse Relaxation		
p	Z^*	Previous Best (Chapter 4)	Simpler Neighbouring Facilities Method[b]	Voronoi Method[b]	Previous Best (Chapter 5)	Simpler Neighbouring Facilities Method[b]	Voronoi Method[b]
10	1716.510	342.78	350.13	343.28	0.78	0.39	0.27
20	1029.715	2856.38	4158.49	4155.32	1.48	1.69	1.05
30	739.193	2146.67	33.12	33.09	2.86	1.84	2.73
40	580.005	1515.29	11.28	11.31	9.97	9.23	8.55
50	468.542	159.49	28.75	28.88	13.79	18.56	11.70
60	400.195	170.38	30.95	30.83	10.00	12.26	11.14
70	357.946	97.63	5.91	5.93	16.77	14.49	13.79
80	312.500	73.52	6.86	6.91	14.95	20.43	16.49
90	280.903	38.01	3.54	3.56	20.83	16.22	13.95
100	256.680	16.77	6.18	6.18	15.55	10.61	13.94
Average	**614.220**	**741.69**	**463.52**	**462.53**	**10.70**	**10.57**	**9.55**

[b] This excludes computational time for the enhanced relaxed p' matheuristic.

Table 6.5: Total Computational Time Spent Finding the Optimal Solution with and without using the Relaxed p' Matheuristic

& 6 respectively. These results are followed by the time spent to optimally solve the problem where the result from the relaxed p' matheuristic using simpler neighbouring facilities method (columns 4 & 7) and the Voronoi-based method (columns 5 & 8) was incorporated into the exact algorithm appropriately.

Table 6.5 shows that incorporating the results found from the relaxed p' matheuristic has decreased the overall computational time required for both exact algorithms on average. Furthermore, the computational time is shortest on average when using the Voronoi-based method. This demonstrates that even if a slightly higher solution value is found using the relaxed p' matheuristic compared to other heuristic methods, the critical points of the solution circles may form a more efficient initial subset of demand points for exact algorithms such as the *ERRA*.

This is an encouraging result, as it means that the relaxed p' heuristic can be used further to help optimally solve larger problems, such as $rat575$, $rat783$, $pr1002$ and $rl1323$. As variant (c) of the changing neighbourhoods techniques paired with the Voronoi facility selection method yielded the best results, this is therefore taken as our overall choice for the relaxed p' matheuristic to use for further analysis with the larger data sets.

6.5.3 Computational Results for Larger Instances

For consistency with Chapters 4 and 5, the relaxed p' matheuristic is also tested on four larger TSP-Library data sets, namely $rat575$, $rat783$, $pr1002$ and $rl1323$, and on the nine generated data sets, see Section 4.9.2, Chapter 4.

Scenario One: Results using the TSP-Library data sets

An initial upper bound, Z_1, was found for the TSP-Library data sets $rat575$, $rat783$, $pr1002$ and $rl1323$ where $p = 10, 20, \ldots, 100$ using 100 iterations of the H_2 heuristic. This was performed so that results remained consistent with that of the data set $pr439$. The initial solution was then used to begin the relaxed p' matheurisitic. This yielded a tight feasible solution and a solution value, $Z^{\perp}$. The critical points of the solution circles then form an initial subset of demand points that was embedded into the $ERRA$ to find the optimal solution, Z^*.

It is important to note that only one exact algorithm is needed to optimally solve the data sets, and the $ERRA$ was chosen due to the quality of the solution values collected from the relaxed p' matheuristic. Although the matheuristic found tight solution values, in general the corresponding Z^H value was tighter. As the Drezner enhanced algorithm has already been implemented using these Z^H values as its starting upper bound for these data sets (see Chapter 4), the $ERRA$ was therefore chosen as the optimal method used to establish if either a) the total computational time could be improved or b) optimal solutions could be found where previously they were unknown.

Tables 6.6-6.9 are arranged in the following way. The first column shows the number of facilities, p, that are required. For comparison, the second column shows the best known heuristic results, Z_H, found in Elshaikh *et al* (2015). The third column shows the initial solution value found using 100 iterations of the H_2 heuristic. The fourth and fifth columns correspond to the solution value, $Z^{\perp}$, and computational time spent for the enhanced relaxed p' matheuristic only respectively. Finally, the sixth column, labelled Z^*, shows the optimal solution found (or best solution found if indicated), and the seventh column shows the respective computational time required. To remain consistent with previous chapters, a time limit of 86400 seconds was set for

each p value. If optimality was not reached in this time, then this is indicated and the best lower bound found is given.

			Relaxed p' Matheuristic		Enhanced Reverse Relaxation Algorithm	
p	Z_H	Z_1	$Z^{\perp}$	CPU Time (secs)[a]	Z^*	CPU Time (secs)[b]
10	67.926	71.491	69.223	33.15	67.926	9.74
20	45.621	53.694	46.744	229.38	45.475	668.75
30	35.556	41.182	36.249	419.98	35.556	1159.08
40	30.265	37.607	30.700	857.64	30.063	85704.40
50	26.173	32.324	27.244	1178.15	25.826	16508.50
60	23.662	30.548	23.880	1017.14	23.163	26356.00
70	21.059	27.117	21.476	1427.56	20.858	32199.70
80	19.510	26.332	19.417	2145.32	19.026	15137.60
90	17.923	24.037	17.805	2013.23	17.460	1415.08
100	16.551	22.738	16.621	2517.60	16.420	7086.69
Average	**30.421**	**36.707**	**30.936**	**1183.92**	**30.178**	**18624.60**

[a] This excludes computational time for the H_2 heuristic.
[b] This excludes computational time for the enhanced relaxed p' matheuristic.

Table 6.6: Results found for $rat575$ using the Relaxed p' Matheuristic

			Relaxed p' Matheuristic		Enhanced Reverse Relaxation Algorithm	
p	Z_H	Z_1	$Z^{\perp}$	CPU Time (secs)[a]	Z^*	CPU Time (secs)[b]
10	79.313	83.474	82.163	18.10	79.313	26.01
20	53.441	59.158	54.002	121.99	53.332	1379.11
30	42.395	49.554	42.852	1204.53	42.307	40613.60
40	35.962	42.244	36.637	2405.47	$35.171^{\perp}$	86400.00
50	31.184	38.791	32.068	1895.61	$30.540^{\perp}$	86400.00
60	28.053	34.714	28.889	2115.47	$27.117^{\perp}$	86400.00
70	25.446	31.520	26.086	2251.02	$24.453^{\perp}$	86400.00
80	23.560	29.488	23.363	2610.17	$22.589^{\perp}$	86400.00
90	21.710	27.122	22.001	4546.07	$20.887^{\perp}$	86400.00
100	20.334	26.420	19.987	5427.31	$19.558^{\perp}$	86400.00
Average	**36.140**	**42.248**	**36.805**	**2259.57**	$---$	**64681.87**

$^{\perp}$ Best lower bound found in 86400.00 secs
[a] This excludes computational time for the H_2 heuristic.
[b] This excludes computational time for the enhanced relaxed p' matheuristic.

Table 6.7: Results found for $rat783$ using the Relaxed p' Matheuristic

For the largest data set (i.e. $n = 1323$), a maximum time limit of 500 seconds was given for each loop. In other words, after 500 seconds the matheuristic automatically diversifies the solution to keep the algorithm as quick and efficient as possible. It is important to note that this condition was only needed for $n = 1323$ where $p > 70$, as the smaller data sets and values of p diversified the solution before taking 500 seconds in each loop.

When comparing to the best known heuristic value, we see that the relaxed p' matheuris-

tic does find tighter upper bound solution values in some instances (e.g. $p = 80$ & 90 for $rat575$), thus obtaining a tighter feasible solution. However, mixed results can be found over the different TSP-Library data sets when embedding the result from the relaxed p' matheuristic into the $ERRA$. Some instances require less computational time to find the optimal solution, such as the data set $pr1002$ where $p = 10, 20, 50, 60$ & 100, which is the desired result. However, the other p values for this data set require more computational time (greatly in some instances), and therefore the average computational time needed has increased. This highlights the $ERRA$'s sensitivity to the initial subset of demand points.

			Relaxed p' Matheuristic		Enhanced Reverse Relaxation Algorithm	
p	Z_H	Z_1	$Z^{\perp}$	CPU Time (secs)[a]	Z^*	CPU Time (secs)[b]
10	2389.36	2799.02	2425.20	13.20	2389.36	15.68
20	1609.54	1771.52	1677.05	51.81	1607.53	1733.07
30	1231.36	1491.08	1284.54	234.15	$1231.25^{\perp}$	86400.00
40	1030.40	1226.02	1029.27	511.02	1021.41	1323.71
50	901.455	1067.06	946.258	320.82	895.342	10092.8
60	801.474	1007.78	827.647	548.24	795.709	6593.11
70	727.154	923.870	732.913	1153.88	725.431	7648.74
80	664.798	864.459	687.841	1224.99	655.746	1819.98
90	604.152	811.327	625.000	1252.83	$603.078^{\perp}$	86400.00
100	559.017	790.569	579.871	1313.05	555.662	330.91
Average	**1051.870**	**1275.271**	**1081.559**	**662.40**	$---$	**20235.80**

$^{\perp}$ Best lower bound found in 86400.00 secs

[a] This excludes computational time for the H_2 heuristic.

[b] This excludes computational time for the enhanced relaxed p' matheuristic.

Table 6.8: Results found for $pr1002$ using the Relaxed p' Matheuristic

Promising results were found for the more challenging data sets (i.e. $rat783$ and $rl1323$), as tighter lower bounds were obtained for the data set $rat783$ in the maximum time limit when $p = 60, 80$ & 100 compared to the results in the previous chapter, and tighter lower bounds were obtained for the data set $rl1323$ in the maximum time limit when $p = 40, 60, 70$ & 80. Furthermore, an optimal solution was found for the first time for the data set $rl1323$ where $p = 100$ (highlighted in bold in Table 6.9).

It is interesting to see that the initial subset of demand points found using the relaxed p' matheuristic appears to be more efficient the larger the data set is. The next section will investigate the performance of the matheuristic further to see if any patterns can be revealed using the generated data sets.

			Relaxed p' Matheuristic		Enhanced Reverse Relaxation Algorithm	
p	Z_H	Z_1	$Z^{\perp}$	CPU Time (secs)[a]	Z^*	CPU Time (secs)[b]
10	2897.49	3059.26	3018.75	17.83	2897.489	973.70
20	1886.82	2046.31	1959.13	118.97	1868.921	1643.50
30	1466.97	1646.39	1494.83	573.68	1466.970	64058.82
40	1236.38	1434.84	1240.62	522.74	$1226.848^{\perp}$	86400.00
50	1060.82	1333.30	1072.45	605.67	$1048.929^{\perp}$	86400.00
60	941.870	1115.00	959.591	1103.64	$931.919^{\perp}$	86400.00
70	844.967	1046.04	872.325	2035.63	$841.976^{\perp}$	86400.00
80	774.764	959.908	785.973	2767.02	$770.753^{\perp}$	86400.00
90	720.625	895.898	736.912	2914.48	706.145	9192.26
100	662.936	843.894	685.596	2226.94	**658.997**	59703.30
Average	**1249.364**	**1438.084**	**1282.618**	**1288.66**	$---$	**56757.16**

$^{\perp}$ Best lower bound found in 86400.00 secs.

[a] This excludes computational time for the H_2 heuristic.

[b] This excludes computational time for the enhanced relaxed p' matheuristic.

Table 6.9: Results found for $rl1323$ using the Relaxed p' Matheuristic

Scenario Two: Results using our new generated data sets

This section displays the results found using the relaxed p' matheuristic for the generated data sets in Tables 6.10-6.12. The first column displays the data's distribution type, which are discussed in Section 4.9.2, Chapter 4. The second column displays the number of facilities located. The third column shows the upper bound found using 100 iterations of the H_2 heuristic, which was used as an initial solution for the relaxed p' matheuristic. The fourth and fifth columns displays the solution value found using the relaxed p' matheuristic and its overall computational time respectively. The sixth and seventh columns display the optimal solution found when embedding the result from the matheuristic into $ERRA$ and the overall computational time respectively. Finally, the eighth column shows how far the solution value found using the enhanced relaxed p' matheuristic deviates from the optimal solution. Note that the deviation was calculated by

$$\frac{Z^{\perp} - Z^*}{Z^*}.$$

Results show that the relaxed p' matheuristic found a tight upper bound on the optimal solution in reasonable time for all values of p and distribution types. It is evident that the matheuristic is most efficient at obtaining a tight upper bound for clustered data sets. If we study the deviation of $Z^{\perp}$ from Z^* we see that the best solutions are

found with the clustered data with an average deviation of 0.008 for all three data sets. Furthermore, there are several instances where the relaxed p' matheuristic found the optimal solution for this distribution type (i.e. the deviation is 0) and for the semi-clustered data.

		H_2 Heuristic	Relaxed p' Matheuristic		Enhanced Reverse Relaxation Algorithm		
Dist	p	Z_1	$Z^{\perp}$	CPU Time (secs)[a]	Z^*	CPU Time (secs)[b]	Deviation
Cl	25	71.586	53.677	137.82	53.677	1.45	0.000
Cl	50	49.189	33.245	334.68	32.660	6.55	0.018
Cl	75	33.971	24.279	649.84	24.279	10.35	0.000
Cl	100	28.055	19.983	974.223	19.981	19.96	0.000
Average	**63**	**45.700**	**32.791**	**574.14**	**32.645**	**6.60**	**0.005**
S-Cl	25	125.071	91.100	24.78	89.098	4.78	0.022
S-Cl	50	82.607	53.891	168.26	53.891	3.50	0.000
S-Cl	75	72.222	38.852	432.37	38.852	7.58	0.000
S-Cl	100	63.118	29.568	601.18	29.568	4.52	0.000
Average	**63**	**85.755**	**53.353**	**306.65**	**52.852**	**5.09**	**0.006**
R	25	132.171	110.637	93.87	107.353	95.80	0.031
R	50	91.800	71.143	286.57	67.151	23.18	0.059
R	75	66.532	53.388	424.29	51.574	15.32	0.035
R	100	59.483	40.577	677.89	40.140	9.72	0.011
Average	**63**	**87.497**	**68.936**	**370.66**	**66.554**	**36.00**	**0.034**

[a] This excludes computational time for the H_2 heuristic.
[b] This excludes computational time for the enhanced relaxed p' matheuristic.

Table 6.10: Results found for the generated data where $n = 400$ using the Relaxed p' Matheuristic

		H_2 Heuristic	Relaxed p' Matheuristic		Enhanced Reverse Relaxation Algorithm		
Dist	p	Z_1	$Z^{\perp}$	CPU Time (secs)[a]	Z^*	CPU Time (secs)[b]	Deviation
Cl	25	74.500	59.010	62.06	57.618	13.50	0.024
Cl	50	44.444	37.620	299.41	36.739	22.18	0.024
Cl	75	38.484	26.077	525.66	25.986	29.03	0.024
Cl	100	41.150	21.101	824.741	20.826	31.83	0.013
Average	**63**	**49.645**	**35.952**	**427.968**	**35.292**	**24.13**	**0.016**
S-Cl	25	81.416	65.324	81.37	64.368	70.30	0.015
S-Cl	50	53.740	42.500	310.96	41.904	161.62	0.014
S-Cl	75	46.406	31.336	624.27	31.275	31.27	0.002
S-Cl	100	40.008	25.224	799.59	24.323	24.32	0.037
Average	**63**	**55.393**	**41.096**	**454.05**	**40.468**	**76.81**	**0.017**
R	25	74.577	70.738	110.61	67.979	1022.72	0.041
R	50	55.009	46.374	460.63	44.411	4166.66	0.044
R	75	46.652	34.515	742.16	33.616	213.12	0.027
R	100	38.916	27.166	1116.57	27.023	55.37	0.005
Average	**63**	**53.789**	**44.698**	**607.49**	**43.257**	**1364.47**	**0.029**

[a] This excludes computational time for the H_2 heuristic.
[b] This excludes computational time for the enhanced relaxed p' matheuristic.

Table 6.11: Results found for the generated data where $n = 600$ using the Relaxed p' Matheuristic

Figure 6.16 displays the deviation for the three distribution types where $n = 400, 600$ & 800. The semi-clustered data appears to deviate more from the optimal solution

as the problem size increases. This trend can be explained by examining how the matheuristic may behave for this distribution type. For a semi-clustered data set, some areas are highly concentrated with demand points and some are sparsely populated. This makes it more likely that the matheuristic will become stuck in a highly populated area as it is unable to incorporate the sparsely distributed demand points into its neighbourhood. Therefore, the matheuristic may need to diversify its solution in order to escape becoming stuck in a local minimum more often. Thus as the problem size grows, the quality of the solution value may decrease.

		H_2 Heuristic	Relaxed p' Matheuristic		Enhanced Reverse Relaxation Algorithm		
Dist	p	Z_1	$Z^{\perp}$	CPU Time (secs)[a]	Z^*	CPU Time (secs)[b]	Deviation
Cl	25	84.465	60.879	193.577	60.879	5.76	0.000
Cl	50	49.218	40.265	811.70	40.265	40.265	0.000
Cl	75	38.604	30.446	2121.07	30.446	120.05	0.000
Cl	100	33.136	25.125	3547.93	24.824	139.57	0.012
Average	**63**	**51.356**	**39.179**	**1668.57**	**39.104**	**105.38**	**0.003**
S-Cl	25	97.572	84.678	99.98	81.238	519.45	0.042
S-Cl	50	70.991	53.866	331.24	51.051	379.27	0.055
S-Cl	75	55.587	38.601	782.99	38.596	100.464	0.000
S-Cl	100	46.335	32.596	990.80	31.205	122.37	0.045
Average	**63**	**67.621**	**52.476**	**551.249**	**50.523**	**280.39**	**0.036**
R	25	140.424	117.183	166.61	115.444	942.97	0.015
R	50	95.784	77.820	827.19	75.935	57548.00	0.025
R	75	74.575	59.161	1181.02	58.217	30230.70	0.016
R	100	68.793	50.447	1858.29	48.545	9881.95	0.039
Average	**63**	**94.894**	**76.153**	**1008.28**	**74.535**	**24650.90**	**0.024**

[a] This excludes computational time for the H_2 heuristic.
[b] This excludes computational time for the enhanced relaxed p' matheuristic.

Table 6.12: Results found for the generated data where $n = 800$ using the Relaxed p' Matheuristic

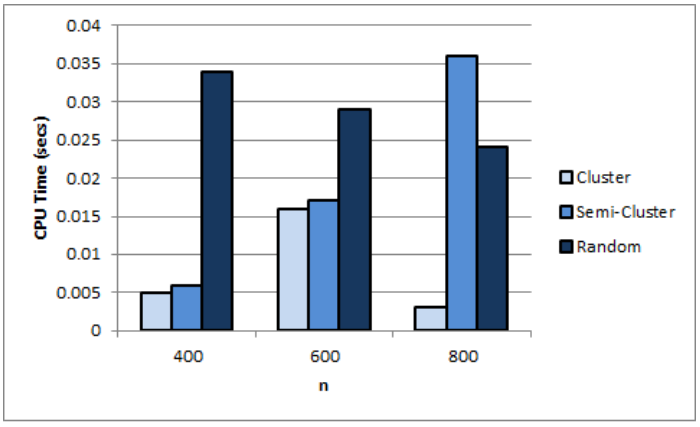


Figure 6.16: Deviation of the relaxed p' matheuristic solution from the optimal solution

Interestingly, the larger the problem size, the closer the relaxed p' matheuristic's solu-

tion is to the optimal solution for the randomly distributed data. This may be because the matheuristic is able to incorporate more demand points into its neighbourhood when the data set is larger as, although the demand points are sparsely distributed, the concentration per unit of space may be higher. Therefore, the matheuristic is able to find a tighter solution value in this instance.

The results for the generated data sets are therefore consistent with our findings for the TSP-Library data sets.

6.6 Summary

This chapter has developed a new matheuristic that was used to optimally solve large data sets from the TSP-Library and nine generated data sets. The matheuristic combined the well-known locate-allocate heuristic with elements from the $ERRA$ proposed in Chapter 5 in order to find a tight upper bound and feasible solution to the $p-$centre problem. This feasible solution was then embedded into the enhanced exact algorithms developed in Chapter 4 and 5 to find the optimal solution. The two most crucial parts to the algorithm consisted of a) selecting a good subset of facilities and b) changing the neighbourhood efficiently. These two vital areas were investigated with several variants in order to establish the most appropriate matheuristic, and the best variant was chosen as the final matheuristic according to the computational time, number of iterations and the overall solution value found. Results show that in some instances, the matheuristic finds a better solution value than the best known heuristic solution, and for the largest TSP-Library data set an optimal solution was obtained for the first time.

The next chapter will adapt the relaxation-based algorithm developed in Chapter 5 to solve related location problems.

Chapter 7

Relaxation-Based Method to Related $p-$Centre Problems: Formulations & Managerial Insights

7.1 Introduction

This chapter will explore the adaptation of the Enhanced Reverse Relaxation Algorithm ($ERRA$) for two variations of the classic $p-$centre problem, namely the $\alpha-$ neighbour $p-$centre problem and the conditional $p-$centre problem. These two related location problems demonstrate the effectiveness and the flexibility of this optimal method. The first section will investigate an adaptation of the $ERRA$ for the $\alpha-$ neighbour $p-$centre problem, followed by experiments where $\alpha = 2$ and 3. For managerial insight, we also conduct a scenario analysis where the coverage demand (i.e. the number of facilities that are required to cover a demand point) varies between the demand points. The second section investigates the conditional $p-$centre problem where a modified algorithm that is based on $ERRA$ is proposed and compared to a recently published approach with encouraging results.

7.2 The $\alpha-$ Neighbour $p-$Centre Problem

7.2.1 Introduction

As previously stated in Section 2.2.5, Chapter 2, the $\alpha-$ neighbour $p-$centre problem aims to minimise the maximum distance between each demand point and its closest facility such that every demand point is covered by at least α facilities (where $\alpha < p$). This is equivalent to ensuring that each demand point is covered by at least α covering circles (see Figure 7.1 for an example where $\alpha = 3$). A strength of this classification type is that it allows for either failure or closure of $\alpha - 1$ facilities whilst ensuring that

each demand point is still covered. This provides extra safety and security, which is particularly important when locating emergency facilities as the closure of a facility could result in a deeply upsetting or damaging outcome if there is no alternative facility that can be relied on. Note that for $\alpha = 1$, the problem reduces to the classical $p-$centre problem.

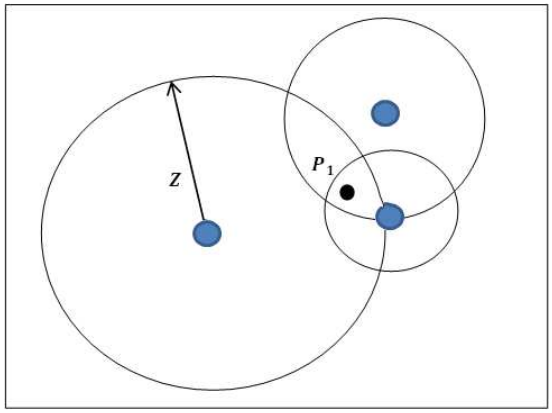


Figure 7.1: Demand point P_1 is covered by 3 facilities for the solution value Z

Chen & Chen (2013) were the first to propose an optimal algorithm for the $\alpha-$neighbour $p-$centre problem. They adapted and compared two exact algorithms, namely Minieka's (1970) algorithm used to solve the discrete and continuous $p-$centre problem, and their earlier classic relaxation algorithm (Chen & Chen (2009)). They showed experimentally that the latter is more efficient. We therefore wish to enhance the efficiency of such an optimal algorithm by adapting $ERRA$.

7.2.2 Adapting $ERRA$ for the $\alpha-$Neighbour $p-$Centre Problem

Chen & Chen (2013) stated the adjustments that need to be made to the classic relaxation algorithm to optimally solve the $\alpha-$neighbour $p-$centre problem. $ERRA$ shares many similar properties to the classic relaxation algorithm, and so some adaptations for $ERRA$ are similar though some are different. Therefore, the four modifications that we propose will be presented next.

1. Changes to the Formulation

The formulation for the classic $p-$centre problem, For_{pc} given in Section 3.2.2, Chapter 3, is modified accordingly for the $\alpha-$ neighbour $p-$centre problem. The modified

formulation, For^{α}_{pc}, is given here.

$$For^{\alpha}_{pc}: \text{Minimise} \quad W \tag{7.1}$$

$$\text{subject to} \quad \sum_{j \in J} A_{i,j} x_j \geq \alpha \quad \forall i \in I, \tag{7.2}$$

$$\sum_{j \in J} x_j = p, \tag{7.3}$$

$$W \geq x_j r_j \quad \forall j \in J, \tag{7.4}$$

$$x_j \in \{0,1\} \; \forall i \in I, \forall j \in J, \tag{7.5}$$

$$W \geq 0, \tag{7.6}$$

where

α: the minimum number of facilities needed to cover a demand point.

All other notations are as previously given in Chapter 3.

The objective function (7.1) minimises the maximum distance between any demand point and facility. Constraint (7.2) guarantees that every customer i is covered by at least α facilities. Constraint (7.3) ensures that only p facilities are located and constraint (7.4) imposes that W is the maximum distance. Constraint (7.5) and constraint (7.6) refer to the binary decision variable and the continuous decision variable respectively.

Note that *ERRA* uses the set covering based formulation rather than the classical $p-$centre problem formulation to solve the subset of demand points optimally. Therefore, the set covering problem, given in Section 3.2.1, Chapter 3, is modified to accommodate the characteristics of the new problem which we refer to as For^{α}_{sc}.

$$For^{\alpha}_{sc}: \text{Minimise} \quad \sum_{j \in J} x_j \tag{7.7}$$

subject to (7.2) and (7.5).

The objective function (7.7) is to minimise the number of open facilities, and the constraints are as given previously.

2. Adjusting the Enhancement $AddE2$

For both the classic relaxation algorithm and $ERRA$, if the solution to the subset is not feasible for the full problem, then k demand points are added to the subset of demand points. In the case of the classic relaxation algorithm, Chen & Chen added the k demand points that lie furthest from the p covering circles. However, after identifying a weakness with this selection method in Chapter 5, we proposed the following method which we shall briefly reiterate here for convenience.

Artificial circles with radii Z_t are constructed around each facility, where Z_t is the solution value at iteration t. The demand points that are not encompassed by their closest artificial circles are identified. This creates at most p clusters of uncovered demand points. The k furthest demand points from these clusters are added to the subset of demand points such that *only one demand point is selected from each cluster.*

In the case of the $\alpha-$ neighbour $p-$centre problem, Chen & Chen adjusted the classic relaxation algorithm so that the k demand points that lie the furthest from their α^{th} nearest facility were added to the subset of demand points instead.

We therefore amend Step 2 in the Point Selection Algorithm (see Figure 5.12, Chapter 5) to form the Adapted Point Selection Algorithm given in Figure 7.2.

1. Step 1 in Figure 5.12.
2. Allocate all the demand points $i \in I$ to their α closest facility $j \in F$ and define the allocation matrix $A_{i,j}$ where $A_{i,j} = 1$ if demand point i is allocated to facility j, else $A_{i,j} = 0$. Record $Z = Max(r_j : \ j \in F)$.
3. Steps $3 - 5$ in Figure 5.12.

Figure 7.2: Adapted Point Selection Algorithm

3. Finding the Solution Value

The $p-$centre problem wishes to minimise the worst case scenario, and therefore the solution value for the classic $p-$centre problem is simply the radius of the largest covering circle. For the $\alpha-$ neighbour $p-$centre, the worst case scenario would be the

failure of the closest $(\alpha - 1)$ facilities serving a demand point. Therefore, the solution value for this problem is also the radius of the largest covering circle.

4. Co-Location of Facilities

Co-location of facilities means that more than one facility to be situated at the same location. This is not a possibility for the classic $p-$centre problem, as the extra facility could be located inside the solution circle to improve the solution value assuming the solution circle is unique. However, this observation does not apply to the $\alpha-$neighbour $p-$centre problem as co-location may be beneficial in some instances.

For clarity, let (x_{i_c}, y_{i_c}) be the Cartesian coordinates of demand point i and (x_{j_c}, y_{j_c}) the Cartesian coordinates of facility j.

Let

$$i' = Arg[\underset{i \in I}{Max}\ \{d_{i,j}\ \forall\ j \in J : (x_{j_c}, y_{j_c}) \neq (x_{i_c}, y_{i_c})\}],$$

where J' is the set of α closest circles to demand point i', and Z^{DP} is the optimal solution value for the $\alpha-$neighbour $p-$centre problem for a given set of demand points, denoted DP. Finally, let $1 < \alpha < p$.

Lemma 1. *If $\exists\ d_{i',j''} > Z^{I\setminus\{i'\}}$ for $j'' \in J'$, then the co-location of facilities will improve the solution value Z^I.*

Proof. We know that the optimal solution, with solution value $Z^{I\setminus\{i'\}}$, for the $\alpha-$neighbour $p-$centre problem covers demand points $I \setminus \{i'\}$. Therefore, Z^I must cover demand point i' with α circles. We also know that at least one $j'' \in J'$ has a distance

$$d_{i',j''} > Z^{I\setminus\{i'\}}$$

to demand point i', which implies

$$Z^I > Z^{I\setminus\{i'\}}.$$

Therefore, in order to reduce Z^I (i.e. $Z^I = Z^{I\setminus\{i'\}}$), p' facilities must be co-located at a facility site $j'' \in J'$ such that $d_{i',j''} \leq Z^{I\setminus\{i'\}}$ where $1 < p' \leq \alpha$. □

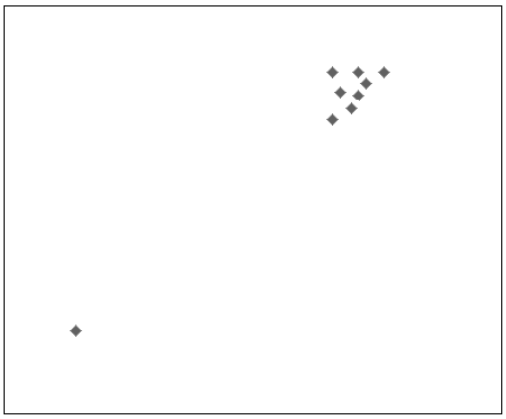
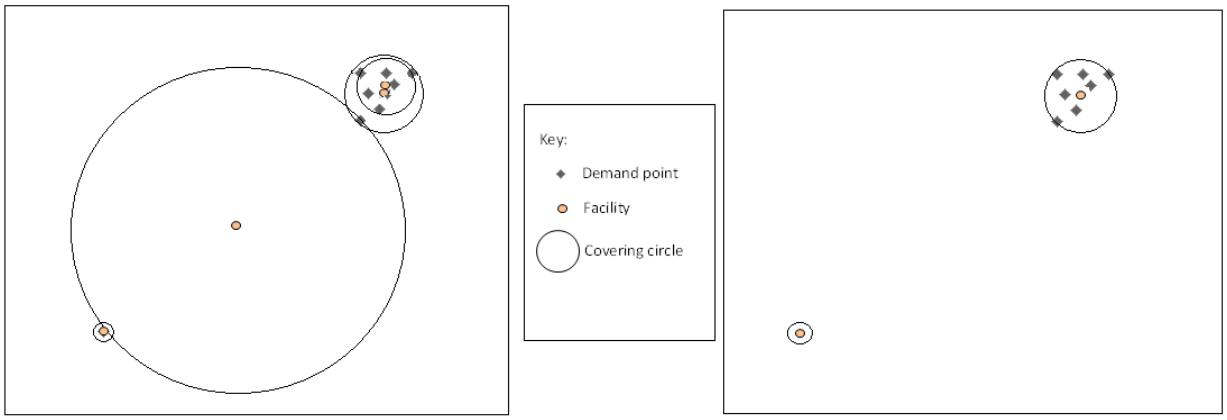


Figure 7.3: A simple set of demand points

A Simple Illustrative Example

Let us demonstrate the usefulness of co-location with a simple example. Figure 7.3 shows a small set of demand points. Figure 7.4a shows the solution to the 2−neighbour 4−centre problem for this data set where co-location is not permitted, yielding a solution value of 6.72. Figure 7.4b displays the solution for the 2−neighbour 4−centre problem for the same data set where co-location is allowed. In this case, two facilities are located at each site yielding an improved solution with value 2.83.



(a) Co-location is not allowed (b) Co-location is allowed

Figure 7.4: Solving the 2−neighbour 4−centre problem

In order to allow co-location of facilities, constraint (7.5) in For_{sc}^{α} was altered to constraint (7.8) where x_j becomes a bounded integer variable instead of a binary one.

$$x_j \in \{0, \alpha\} \quad \forall j \in J. \tag{7.8}$$

It is worth noting that this new constraint does not affect the solution value found using For_{sc}^{α}.

The four adaptations were incorporated into $ERRA$ to form the Adapted Enhanced Reverse Relaxation Algorithm ($AERRA$), which is given in Figure 7.5 for completeness.

1. Input: α. Set the lower bound $LB = 0$, $c = 1$, $t = 0$ and $jump_{Max} = 10$.
2. Select the initial subset of demand points using Initial Subset Algorithm given in Figure 5.5.
3. Set $t = t+1$. Determine if Sub has a feasible solution for the $\alpha-$neighbour $p-$centre problem, $feasible$, with a solution value, $Z_t \leq LB$.
   a) If $feasible$ cannot be found, determine the value of $jump$ using Equation (5.11) and find the $jump^{th}$ smallest lower bound, LB_{jump}, that is larger than LB. Set $LB = LB_{jump}$ and go back to the beginning of Step 3.
   b) If $feasible$ can be found, find the number of uncovered demand points, N_u^t, and continue to Step 4.
4. Determine whether $feasible$ is feasible for the full problem.
   a) If it is, determine whether Sub has a feasible solution, $feasible$, with a solution value $\leq LB_{(jump-c)}$.
      If not, return $LB_{jump-(c-1)}$ as the final solution value.
      Else set $c = c + 1$ and repeat Step 4(a).
   b) Determine the value of k using Equation (5.12). Add k demand points to Sub using the Adapted Point Selection Algorithm given in Figure 7.2 and go to Step 3.

Figure 7.5: The Adapted Enhanced Reverse Relaxation Algorithm ($AERRA$)

7.2.3 Computational Results

In this section, we first present the results of $AERRA$ for $n = 439$ and $\alpha = 2$ where previous results exist, followed by $\alpha = 3$ where no published results are available. We then test the method further on larger instances where the optimal results for the $\alpha-$neighbour $p-$centre problem for the TSP-Library data sets $rat575$, $rat783$, $pr1002$ and $rl1323$ are produced for the first time.

a) Comparing $AERRA$ to existing results

$AERRA$ optimally solved the $\alpha-$neighbour $p-$centre for the TSP-Library data set $pr439$ where $p = 10, 20, \ldots, 100$ and $\alpha = 2$. For comparison purposes, the results in

Table 7.1 are also given alongside Chen & Chen's (2013) results.

		Classic Relaxation (Chen & Chen)			*AERRA*		
p	Z^*	CPU (secs)	Sub_{Max}	# Iterations	CPU (secs)	Sub_{Max}	# Iterations
10	2752.639	0.37	52	82	0.37	35	24
20	1716.510	3.04	108	162	0.84	60	37
30	1271.830	18.32	158	237	0.94	75	35
40	1008.170	27.20	196	314	3.40	120	59
50	874.271	605.65	250	389	4.53	125	63
60	739.193	978.71	260	404	7.90	154	78
70	621.742	1888.61	306	493	25.81	193	104
80	580.005	1576.88	322	515	24.80	210	112
90	530.477	1737.54	341	565	49.18	247	140
100	463.175	1443.72	352	587	26.89	234	115
Average	**1055.801**	**828.004**	**235**	**375**	**14.52**	**145**	**77**

Table 7.1: Results for the 2−neighbour $p-$centre problem for $pr439$ using $AERRA$

Table 7.1 is structured in the following way. The first column states the number of facilities located, p. The second column, titled Z^*, shows the optimal solution found for the corresponding p value. The next three columns show Chen & Chen's results, with the third column referring to the total amount of computational time required, the fourth to the maximal size of the relaxation sub-problem (Sub_{Max}) and the fifth to the number of sub-problems (iterations) solved. The next three columns are ordered identically, except the results correspond to those found using $AERRA$.

Results show that $AERRA$ found the optimal solutions for the 2−neighbour $p-$centre problem for the data set $pr439$ more efficiently compared to the adapted classic relaxation algorithm used by Chen & Chen. $AERRA$ was found to be superior as it required 98.25% less computational time, 38.30% fewer demand points in the subset and 79.47% fewer iterations to solve the problem optimally. These encouraging results show that $ERRA$ is both reliable and efficient when adapted to optimally solve this related $p-$centre problem.

$AERRA$ was tested further for the data set $pr439$ with $\alpha = 3$ where no published results are available. Table 7.2 shows the number of facilities located, the optimal results found, the total computational time spent, the maximal size of the subset of demand points, and the number of iterations required. Some instances show that less computational time is required for this larger α value (i.e. when $p = 40, 50, 60, 70$ &

	$\alpha = 3$			
p	Z^*	CPU (secs)	Sub_{Max}	# Iterations
10	3989.302	0.49	29	34
20	2347.505	6.27	76	114
30	1716.510	2.01	61	93
40	1407.624	3.21	98	91
50	1226.020	3.23	93	81
60	1019.986	5.76	110	150
70	946.457	17.48	141	215
80	853.028	54.00	172	275
90	739.193	21.18	165	211
100	657.885	77.69	186	384
Average	**1490.35**	**19.05**	**113**	**164**

Table 7.2: Results for the 3−neighbour $p-$centre problem for $pr439$ using $AERRA$

90) when compared to the results for $\alpha = 2$. This may be because the number of demand points needed in the subset has decreased due to the solution value for the $(\alpha+1)-$neighbour $p-$centre problem being either the same or higher than the solution value for the $\alpha-$neighbour $p-$centre problem (this observation will be discussed next). However, for other values of p, the required computational time has increased. A possible reason for this could be as the value of α increases, the solution value becomes larger and so the problem size becomes bigger.

An Observation & Sensitivity Analysis

Let Z_α denote the solution value to the $\alpha-$neighbour $p-$centre problem. As first noted by Chen & Chen (2013), we observe that $Z_\alpha \geq Z_{\alpha-1}$ as the coverage need for each demand point has increased. In other words, if at least one demand point is not covered by α facilities, then at least one covering circle will need to increase in size in order to establish a feasible solution. Figure 7.6 demonstrates this observation for the data set $pr439$ where $p = 10, \ldots, 50$ and $\alpha = 1, 2$ and 3.

We can conclude that $Z_{\alpha-1}$ can be used as an initial lower bound when solving the $\alpha-$neighbour $p-$centre problem.

Furthermore, Table 7.3 shows the percentage increase in the solution value, and therefore additional coverage required, as the value of α increases. In this particular instance, there is a dramatic increase in the solution value on average when $\alpha = 3$ compared to

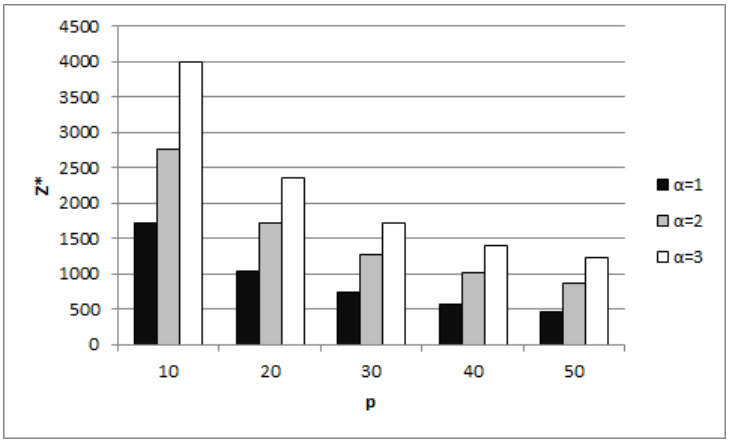


Figure 7.6: The optimal solution value, Z^*, for the data set $pr439$ where $p = 10, \ldots, 50$ and $\alpha = 1, 2$ & 3.

	α=1	$\alpha = 2$		$\alpha = 3$	
p	Z^*	Z^*	% Increase	Z^*	% Increase
10	1716.51	2752.639	58.99	3989.302	132.41
20	1029.715	1716.510	66.70	2347.505	127.98
30	739.193	1271.830	72.06	1716.51	132.21
40	580.005	1008.17	73.82	1407.624	142.69
50	468.542	874.271	86.59	1226.02	161.67
Average	**906.793**	**1574.684**	**71.632**	**2137.392**	**139.39**

Table 7.3: Sensitivity analysis when solving the $\alpha-$neighbour $p-$centre problem where $n = 439$ and $\alpha = 1, 2$ & 3

$\alpha = 1$. From a managerial perspective, this means it may be more desirable to find a balance between reducing the value of α as much as possible whilst ensuring that all demand points have the required coverage. This compromise introduces an alternative setup to the $\alpha-$neighbour $p-$centre problem which will be discussed in the next section.

b) Computational results for the larger data sets

For consistency with the previous chapters, the $\alpha-$neighbour $p-$centre problem was also solved for the TSP-Library data sets $rat575$, $rat783$, $pr1002$ and $rl1323$ where $\alpha = 2$ & 3. The optimal results are shown for the first time in Tables 7.4-7.7. Furthermore, a maximum time limit of 86400 seconds was given for each value of p in order to remain consistent with previous experiments in this thesis.

Results show that $AERRA$ can yield optimal solutions for the large TSP-Library data sets, however the process can be slow. Although there is some evidence to suggest that the required computational time is slightly less for larger values of α, optimal solutions

could not be obtained in the time limit for many instances where $p > 40$.

	$\alpha = 2$				$\alpha = 3$			
p	Z^*	CPU (secs)	Sub_{Max}	# Iterations	Z^*	CPU (secs)	Sub_{Max}	# Iterations
10	114.064	13.29	71	55	137.023	1.77	41	37
20	67.926	110.95	118	111	89.885	67.00	116	80
30	$54.073^{\perp}$	86400.00	235	287	67.926	87.72	121	158
40	$44.782^{\perp}$	86400.00	258	295	58.592	15674.13	238	468
50	$39.061^{\perp}$	86400.00	269	267	$49.941^{\perp}$	86400.00	245	483
60	$34.599^{\perp}$	86400.00	274	235	$44.406^{\perp}$	86400.00	228	363
70	$31.099^{\perp}$	86400.00	281	171	$40.736^{\perp}$	86400.00	261	508
80	$28.513^{\perp}$	86400.00	298	183	$37.207^{\perp}$	86400.00	276	527
90	$26.727^{\perp}$	86400.00	335	221	$34.558^{\perp}$	86400.00	295	518
100	$25.008^{\perp}$	86400.00	346	198	$32.070^{\perp}$	86400.00	284	426
Average	$---$	**69132.42**	**249**	**202**	$---$	**53423.06**	**210**	**356**

$^{\perp}$ Best lower bound found in 86400.00 secs

Table 7.4: Results for the $\alpha-$neighbour $p-$centre problem for $rat575$ and $\alpha = 2$ & 3

	$\alpha = 2$				$\alpha = 3$			
p	Z^*	CPU (secs)	Sub_{Max}	# Iterations	Z^*	CPU (secs)	Sub_{Max}	# Iterations
10	131.846	20.84	79	73	160.721	0.87	33	32
20	79.312	87.30	121	118	106.304	171.29	126	118
30	$63.343^{\perp}$	86400.00	247	373	79.313	248.15	139	202
40	$52.131^{\perp}$	86400.00	268	314	$69.065^{\perp}$	86400.00	236	796
50	$45.868^{\perp}$	86400.00	273	238	$58.831^{\perp}$	86400.00	257	898
60	$40.336^{\perp}$	86400.00	302	264	$51.886^{\perp}$	86400.00	246	1073
70	$36.433^{\perp}$	86400.00	283	173	$47.383^{\perp}$	86400.00	249	860
80	$33.634^{\perp}$	86400.00	318	198	$43.276^{\perp}$	86400.00	254	747
90	$31.117^{\perp}$	86400.00	321	154	$40.083^{\perp}$	86400.00	295	773
100	$29.300^{\perp}$	86400.00	352	165	$37.314^{\perp}$	86400.00	270	632
Average	$---$	**69130.81**	**256**	**207**	$---$	**60522.03**	**210**	**613**

$^{\perp}$ Best lower bound found in 86400.00 secs

Table 7.5: Results for the $\alpha-$neighbour $p-$centre problem for $rat783$ and $\alpha = 2$ & 3

For the instances where the time limit was reached, it is important to note that an optional extra step could be completed to find a feasible, but not necessarily optimal, solution. This can be achieved by embedding the current locations of the p covering circles (i.e. the facilities) found using $AERRA$ into a heuristic, such as Cooper's locate-allocate heuristic, as the initial facility locations in order to yield an upper bound, and therefore a feasible solution. As a small example, the data sets $rat575$ and $rl1323$ were re-run for $p = 100$, and the last locations of the facilities found using the $AERRA$ were saved. All demand points were allocated to their α closest saved facility where $\alpha = 2$.

This yielded the upper bound values 31.085 & 1205.85 for the 2-neighbour $p-$centre problem for $rat575$ and $rl1323$ respectively. Due to time restrictions, as well as the current focus being on optimal methods, this could not be performed on all data sets in this study, however this method is worth considering and exploring for future research.

	$\alpha = 2$				$\alpha = 3$			
p	Z^*	CPU (secs)	Sub_{Max}	# Iterations	Z^*	CPU (secs)	Sub_{Max}	# Iterations
10	3641.566	1.76	51	34	5268.064	2.96	45	39
20	2389.356	193.89	128	127	3107.089	21.83	93	75
30	1916.540	72557.25	262	396	2389.356	123.40	131	125
40	1568.679$^{\perp}$	86400.00	268	243	2038.961	45414.38	260	385
50	1347.869	3525.44	293	217	1760.26$^{\perp}$	86400.00	251	328
60	1221.289$^{\perp}$	86400.00	332	249	1563.75$^{\perp}$	86400.00	292	301
70	1110.214$^{\perp}$	86400.00	351	247	1430.211$^{\perp}$	86400.00	307	311
80	1011.461$^{\perp}$	86400.00	420	267	1290.213$^{\perp}$	86400.00	302	233
90	940.597$^{\perp}$	86400.00	383	232	1221.17$^{\perp}$	86400.00	357	276
100	866.083$^{\perp}$	86400.00	360	176	1135.70$^{\perp}$	86400.00	336	240
Average	$---$	**59467.83**	**285**	**218**	$---$	**56396.26**	**237**	**231**

$^{\perp}$ Best lower bound found in 86400.00 secs

Table 7.6: Results for the $\alpha-$neighbour $p-$centre problem for $pr1002$ and $\alpha = 2$ & 3

	$\alpha = 2$				$\alpha = 3$			
p	Z^*	CPU (secs)	Sub_{Max}	# Iterations	Z^*	CPU (secs)	Sub_{Max}	# Iterations
10	4441.013	8.56	67	52	6200.039	2.85	46	39
20	2882.010$^{\perp}$	86400.00	237	433	3716.203	40.39	144	110
30	2222.236$^{\perp}$	86400.00	259	452	2880.940$^{\perp}$	86400.00	252	430
40	1827.67$^{\perp}$	86400.00	279	471	2372.675$^{\perp}$	86400.00	264	662
50	1587.654$^{\perp}$	86400.00	299	335	2090.673$^{\perp}$	86400.00	287	783
60	1428.493$^{\perp}$	86400.00	324	321	1809.207$^{\perp}$	86400.00	264	675
70	1275.304$^{\perp}$	86400.00	335	213	1648.718$^{\perp}$	86400.00	293	935
80	1173.849$^{\perp}$	86400.00	378	252	1503.291$^{\perp}$	86400.00	296	637
90	1087.53$^{\perp}$	86400.00	365	216	1417.600$^{\perp}$	86400.00	293	693
100	1009.66$^{\perp}$	86400.00	394	205	1326.82$^{\perp}$	86400.00	334	807
Average	$---$	**77760.86**	**294**	**297**	$---$	**69124.32**	**247**	**577**

$^{\perp}$ Best lower bound found in 86400.00 secs

Table 7.7: Results for the $\alpha-$neighbour $p-$centre problem for $rl1323$ and $\alpha = 2$ & 3

As previously mentioned, a more promising approach with an alternative setup for the $\alpha-$neighbour $p-$centre problem will be investigated in the next section.

7.2.4 The Variable $\alpha-$neighbour $p-$centre Problem

It may be beneficial from a managerial view to investigate the case where the number of times a demand point needs to be covered by a facility varies. In other words, not every demand point needs to be covered by exactly α facilities, but rather each may require α_i facilities covering them.

There are many real life cases where the coverage for one demand point may be more vital than the coverage for another. For example, if a demand point represents a school, it may be more important to locate several hospitals nearby in case one of the allocated hospitals fails or closes. Furthermore, demand points may be ranked by their importance; the more important the demand point is, the more facilities should cover it. This problem has been considered for other types of location problem. For example, Church & Gerrard (2003) investigated the multi-level set covering problem where the number of facilities covering a demand point varied. For the case of the $p-$centre problem, we shall name this problem the variable $\alpha-$neighbour $p-$centre problem.

As each demand point i is now assigned a specific α value, say α_i, constraint (7.2) in For_{sc}^{α} is altered, and constraint (7.5) is also adjusted. For clarity, the full formulation of this variable $\alpha-$neighbour $p-$centre problem, $For_{sc}^{\alpha_{var}}$, is shown below.

$For_{sc}^{\alpha_{var}}$: Minimise
$$\sum_{j \in J} x_j \tag{7.9}$$

subject to
$$\sum_{j=0}^{m} A_{i,j} x_j \geq \alpha_i \quad \forall\, i \in I, \tag{7.10}$$
$$x_j \in \{0, \alpha_{Max}\} \quad \forall\, j \in J, \tag{7.11}$$

where

α_i is the required number of times demand point i needs to be covered
(i.e., $1 \leq \alpha_i \leq \alpha_{Max}$).

α_{Max} being the maximum number of times any demand point needs to be covered.

All other definitions are as previously given.

The objective function (7.9) minimises the number of facilities located. Constraint

(7.10) ensures that every demand point i is covered by at least the required α_i facilities, and constraint (7.11) represents the integer decision variable x_j.

Defining the α_i Value

There are many ways the coverage demand for each demand point may be constructed. For example, the data could be partitioned into clustered, semi-clustered and sparsely distributed regions. The demand points in the more clustered areas could be assigned higher α values to accommodate for the higher concentration of demand. Furthermore, each demand point could also be assigned its α value based on its importance. In other words, the α value could be based on the demand point's importance ranking (i.e. high, medium or low).

However, for simplicity, and to provide the possibility for other researchers to conduct similar experiments, in our investigation we allocated the α_i value for each demand point based on its position in the data set. In other words, for each demand point $i \in I$ we set

$$\alpha_i = \begin{cases} 1 & \text{if } \ i \bmod 3 = 1, \\ 2 & \text{if } \ i \bmod 3 = 2, \\ 3 & \text{if } \ i \bmod 3 = 0. \end{cases} \tag{7.12}$$

This therefore demonstrates a simple example where the demand points are assigned one of three potential α values.

Computational Results for the variable $\alpha-$neighbour $p-$centre problem

The variable $\alpha-$neighbour $p-$centre problem was tested on the TSP-Library data sets $pr439$, $rat575$, $rat783$, $pr1002$ & $rl1323$. The results are given in Tables 7.8-7.12 which are structured as follows. The first column shows the number of facilities located, p, and the second column displays the optimal solution value, Z^*. The last three columns show the total computational time in seconds, the maximum number of demand points in the subset, Sub_{Max}, and the total number of iterations respectively. For consistency with previous results, a maximum time limit of 86400 seconds was given to optimally solve each data set for each p value. If an optimal solution could not be found then

the best lower bound for the optimal solution obtained is shown.

It can be observed that optimal results were obtained in much less computational time compared to the classic $\alpha-$neighbour $p-$centre problem. An example can be seen when studying the data set $rat575$. When solving the 3$-$neighbour $p-$centre problem for this data set, optimal solutions could not be found in the time limit of 86400 seconds for $p = 50, 60, \ldots, 100$. However, when solving the variable $\alpha-$neighbour $p-$centre problem, optimal solutions were found for all values of p with an average overall computational time of only 2209.90 seconds. Furthermore, optimal solutions were obtained for the most challenging data sets (i.e. $rat783$ & $rl1323$). This is a positive result, especially from a managerial perspective, as varying the number of facilities to cover each demand point may be a more realistic and desirable option.

p	Z^*	CPU (secs)	Sub_{Max}	# Iterations
10	3855.779	1.41	49	67
20	2298.267	1.87	68	94
30	1636.784	2.03	87	94
40	1381.179	4.75	110	142
50	1125.972	3.40	115	81
60	936.833	8.85	148	154
70	850.827	13.91	151	245
80	748.853	11.88	168	177
90	637.377	20.72	205	170
100	586.090	15.70	198	154
Average	**1405.796**	**8.45**	**129**	**137**

Table 7.8: Results for the variable $\alpha-$neighbour $p-$centre problem for $pr439$

p	Z^*	CPU (secs)	Sub_{Max}	# Iterations
10	131.787	6.06	45	67
20	88.996	18.30	92	88
30	64.979	79.24	129	112
40	55.322	341.73	178	179
50	46.507	531.97	209	165
60	41.599	914.78	246	187
70	38.108	779.68	225	255
80	34.567	653.07	260	190
90	31.847	16573.05	345	257
100	29.904	2200.61	302	231
Average	**56.362**	**2209.90**	**206**	**170**

Table 7.9: Results for the variable $\alpha-$neighbour $p-$centre problem for $rat575$

p	Z^*	CPU (secs)	Sub_{Max}	# Iterations
10	157.728	21.01	59	83
20	105.643	26.03	94	82
30	76.820	199.37	141	177
40	65.526	1423.54	200	260
50	55.626	5404.38	267	261
60	49.323	33549.65	296	342
70	$44.769^{\perp}$	86400.00	309	268
80	40.812	11710.84	366	253
90	$37.962^{\perp}$	86400.00	373	252
100	35.471	34130.75	440	265
Average	$---$	**25926.56**	**255**	**224**

$^{\perp}$ Best lower bound found in 86400.00 secs

Table 7.10: Results for the variable $\alpha-$neighbour $p-$centre problem for $rat783$

p	Z^*	CPU (secs)	Sub_{Max}	# Iterations
10	5250.952	2.23	43	50
20	2962.790	86.57	106	116
30	2324.059	48.27	127	83
40	1948.236	1010.83	200	190
50	1676.313	6813.59	257	236
60	1494.094	15979.66	320	285
70	1367.708	15070.05	325	277
80	1226.020	5784.52	356	201
90	1155.231	7296.74	363	237
100	1068.878	2628.73	352	189
Average	**2047.428**	**5472.12**	**244**	**186**

Table 7.11: Results for the variable $\alpha-$neighbour $p-$centre problem for $pr1002$

p	Z^*	CPU (secs)	Sub_{Max}	# Iterations
10	6097.429	0.98	37	21
20	3677.728	345.055	132	131
30	2789.975	9415.31	209	373
40	2313.391	31059.45	280	378
50	2022.358	23532.27	299	379
60	1776.000	31290.30	348	432
70	$1604.88^{\perp}$	86400.00	303	418
80	$1464.19^{\perp}$	86400.00	318	310
90	$1351.72^{\perp}$	86400.00	309	243
100	$1264.73^{\perp}$	86400.00	370	295
Average	$---$	**44124.34**	**261**	**298**

$^{\perp}$ Best lower bound found in 86400.00 secs

Table 7.12: Results for the variable $\alpha-$neighbour $p-$centre problem for $rl1323$

7.3 The Conditional $p-$centre Problem

In this section, we will adapt $ERRA$ to optimally solve another related location problem to the $p-$centre problem, namely the conditional $p-$centre problem.

7.3.1 Introduction

The conditional $p-$centre, or $(p,q)-$centre, problem wishes to locate p facilities such that q facilities already exist. When locating emergency facilities, it may be unrealistic to wish to locate p new emergency facilities amongst a set of demand points whilst assuming that there are no existing facilities that need to be accounted for. Therefore, a strength of the conditional $p-$centre problem is that it addresses the real data more effectively, and therefore provides a more accurate and efficient solution value.

This section will enhance an existing optimal algorithm for the $(p,q)-$centre problem by incorporating $ERRA$, as well as adding two enhancements that were inspired by $ERRA$. The existing algorithm, proposed by Chen & Chen (2010), shall be described first, followed by the proposed enhancements. The two algorithms' performance will be compared to highlight areas of strength, as well as areas for possible improvement. The efficiency of the enhanced algorithm will then be demonstrated by optimally solving the $(p,q)-$centre problem where $q = 20$ for the large TSP-Library data sets.

7.3.2 The Algorithms

Chen & Chen's Algorithm

As briefly described in Chapter 2, Drezner (1989) proposed an optimal algorithm to solve the conditional $p-$centre problem. First, the set of demand points are allocated to their nearest $\varrho \in Q$, where Q is the set of existing facilities. The demand points, and their corresponding nearest distances to the existing facilities, are then sorted into descending order of Euclidean distance. This yields an ordered set of demand points ($OrdI$) and ordered distance vector ($OrdD$). A bisection algorithm is then used to solve a succession of sub-problems until the solution value for the subset is greater than or equal to the next furthest distance in the list.

Chen & Chen (2010) incorporated their reverse relaxation algorithm into Drezner's algorithm to optimally solve the sub-problems. They began by creating $OrdI$ and $OrdD$ as suggested by Drezner. The subset of demand points was then solved using the reverse relaxation algorithm, where the subset initially consisted of the first demand point only in $OrdI$. If the solution value for the subset exceeded the next furthest distance in $OrdD$, then all the demand points are covered by this solution value and so the optimal solution has been found. Otherwise, the next demand point in the ordered list was added to the subset and the process was repeated. Chen & Chen's algorithm for the conditional $p-$centre problem is described in Figure 7.7 and will be referred to as $CON\ CCA$.

1. Input: The set of existing facilities Q.
2. Set the lower bound, LB=0, the subset of demand points $Sub = \emptyset$ and $k = 1$.
3. Allocate all $i \in I$ to their nearest $\varrho \in Q$. Sort the demand points in descending order of Euclidean distance. Let $OrdD = \{M_1, M_2, \ldots, M_n\}$ be the set or ordered distances, and $OrdI = \{i'_1, i'_2, \ldots, i'_n\}$ be the set of ordered demand points.
4. Add i'_k to Sub.
5. Determine if there is a feasible solution, with solution value Z^k, for the $p-$centre problem for Sub where $Z^k \leq LB$.
6. If so, determine whether $Z^k \geq M_{k+1}$.

   i) If so, determine if $Z^k < M_k$.
   If so, return Z^k as the optimal solution value and stop.
   Else, return $Max(Z^{k-1}, M_k)$ as the optimal solution value and stop.

   ii) Else, set $k = k + 1$ and go back to Step 4.

   Else, set LB as the smallest radius larger than the current LB and go back to Step 5.

Figure 7.7: Chen & Chen's (2010) algorithm for the conditional $p-$centre ($CON\ CCA$)

Note that in $CON\ CCA$, the optimal solution value is determined in Step 5. It is important to observe that once $Z^k \geq M_{k+1}$, either Z^k, Z^{k-1} or M_k can be determined as the optimal solution value, which we will now demonstrate with a simple example.

Let $M_1 = 8$, $M_2 = 5, M_3 = 3$ and $M_4 = 2$, and let $Z^1 = 0$, $Z^2 = 2$ and $Z^3 = 4$. We can observe that $CON\ CCA$ will find $Z^k \geq M_{k+1}$ when $k = 3$. In other words, when the subset consists of the first three demand points in $OrdI$, the solution value for the $p-$centre problem for this subset is larger than the next largest distance in $OrdD$, and so all the demand points are covered with the solution value 4. However, further analysis must be carried out in order to determine the optimal solution value. We observe that $Z^3 > M_3$. Therefore, we must check where to 'split' the set of demand points such that we can obtain the minimum solution value required to cover both partitions. If we 'split' the set of demand points at demand point k, the first k demand points in $OrdI$ will be covered by the new facilities with a maximum solution value 4, and the remaining demand points will be covered by the existing facilities with a maximum solution value 2. This would make the overall solution value 4. However, this can be improved if we 'split' the subset of demand points at the $(k-1)^{th}$ position, as the first $(k-1)$ demand points in $OrdI$ would be covered by the new facilities with a maximum solution value 2, and the remaining demand points would be covered by the existing facilities with a maximum solution value 3. This option therefore yields an overall, and optimal, solution value 3. Therefore, if $Z^k \geq M_k$, $CON\ CCA$ returns $Max(Z^{k-1}, M_k)$ as the optimal solution.

We shall now discuss how $ERRA$ was incorporated into Chen & Chen's algorithm (Figure 7.7) in order to speed up such an optimal algorithm. Furthermore, two further enhancements that were inspired by $ERRA$ will also be discussed.

Enhancing Chen & Chen's Algorithm

1. Incorporating ERRA

For small problems, the $CON\ CCA$ is very efficient. However, as the problem size increases, so does $|Sub|$. This means large subsets are required to be solved optimally, which leads to computational issues (e.g. memory issues) due to the problem size. We therefore propose incorporating $ERRA$ into the $CON\ CCA$ to optimally solve the subset, with an aim to create a more efficient algorithm that can solve larger problems due to the use of the relaxation method.

2. Enhancement One: An Efficient Initial Sub

The initial Sub in the $CON\ CCA$ consists of a single demand point. However, this can be improved by adding a further $(p-1)$ demand points, as there must be at least this number of demand points in Sub to find a feasible solution for the $p-$centre problem. Thus by adding them straight away, at least $(p-1)$ iterations are spared. We therefore set the initial $Sub = i'_1, i'_2, \ldots, i'_p$ where $i' \in OrdI$.

3. Enhancement Two: Adding more than One Demand Point to Sub

Each time feasibility cannot be found for the full problem, the $CON\ CCA$ adds one demand point only to Sub and repeats the process again. This approach is effective on smaller data sets, but as the problem size increases, so does the final cardinality of Sub. Therefore, the process of adding one demand point at a time yields many iterations until the optimal solution is attained, which greatly increases the total amount of computational time and effort required to solve the problem. However, the algorithm's efficiency can be improved by applying Chen & Chen's (2009) approach of adding more than one demand point to the subset at a time. In other words, instead of adding the next demand point only in the ordered list to Sub, the next k' demand points are added where $k' > 1$.

A dynamic scheme to determine the value of k', similar to PointE4 proposed in Chapter 5, could be investigated for the $CON\ CCA$. However, as a simple demonstration that provides a foundation for further research, in this study we added the next two demand points in the ordered list to Sub (i.e. $k' = 2$). This means that the algorithm has fewer iterations and so can secure the final subset of demand points faster.

In order to obtain the optimal solution value, first either Z^k, M_k or Z^{k-1} (where Z^{k-1} is the solution value for the subset of demand points $Sub \setminus \{i_k\}$) is saved as the potential optimal solution value, *optimal*, after the analysis described in the previous example. Then the missed solution value, Z^{k-1}, must be analysed further. If $Z^{k-1} < M_k$, then *optimal* is the optimal solution value. However, if $Z^{k-1} \geq M_k$, then the algorithm investigates if $Z^{k-1} < M_{k-1}$. If so, Z^{k-1} is the optimal solution, else $Max(Z^{k-2}, M_{k-1})$ is the optimal solution.

1. Input: The set of existing facilities Q.
2. Set the subset of demand points $Sub = \emptyset$, and binary tracker $TAG = 0$.
3. Allocate all $i \in I$ to their nearest $\varrho \in Q$. Sort the demand points in descending order of Euclidean distance. Let $OrdD = \{M_1, M_2, \ldots, M_n\}$ be the set or ordered distances, and $OrdI = \{i'_1, i'_2, \ldots, i'_n\}$ be the set of ordered demand points.
4. Set $Sub = \{i'_1, \ldots, i'_p\}$ and $k = p$.
5. Solve the $p-$centre problem for Sub using the Enhanced Reverse Relaxation algorithm (Chapter 5, Figure 5.16). This yields solution value Z^k.
6. If $TAG = 1$:
   If $Z^k \geq M_{k+1}$.
   Determine if $Z^k < M_k$.
   If so, return Z^k as the optimal solution value and stop.
   Else, return $Max(Z^{k-1}, M_k)$ as the optimal solution value and stop.
   Else return *optimal* as the optimal solution value and stop.
   If $TAG = 0$:
   If $Z^k \geq M_{k+1}$.
   Determine if $Z^k < M_k$.
   If so, set Z^k as *optimal*.
   Else, set $Max(Z^{k-1}, M_k)$ as *optimal*.
   Set $Sub = Sub \setminus \{i'_k\}$, $TAG = 1$, $k = k - 1$ and go back to Step 5.
   Else, set $k = k + 2$, $Sub = Sub \cup \{i'_{k-1}, i'_k\}$ and go back to Step 5.

Figure 7.8: The *CON ERRA*

For clarity, the enhanced algorithm, *CON ERRA*, that incorporates *ERRA* and the two enhancements given above is provided in Figure 7.8.

7.3.3 Computational Results

In this section. we will compare the performance of the two algorithms *CON CCA* and *CON ERRA* in terms of computational time, subset size and number of iterations to determine which is the most efficient.

The $(p, q)-$centre problem was solved using both the *CON CCA* and the *CON ERRA* for the TSP-Library data set *pr*439. Tables 7.13 shows the results for both algorithms

where $q = 10$ & 20. The first column shows the number of new facilities located, p. For comparison purposes, the maximum total number of facilities was 100 (i.e. $p+q \leq 100$). The next three columns correspond to the results for $q = 10$, where the second column displays the corresponding optimal solution, Z^* and columns three and four represent the total computational time in seconds for the *CON CCA* and the *CON ERRA* respectively. The remaining three columns display the same information, but referring to the results where $q = 20$.

	$q = 10$			$q = 20$		
p	Z^*	CPU (secs)		Z^*	CPU (secs)	
		CON CCA	*CON ERRA*		*CON CCA*	*CON ERRA*
10	1429.434	1.83	11.41	981.150	0.79	2.10
20	958.188	2156.90	64.28	880.696	4.67	4.29
30	655.016	4231.22	138.42	705.780	212.90	40.23
40	558.038	9475.67	167.67	545.722	1499.01	91.78
50	439.638	3620.33	229.09	445.939	1904.46	171.03
60	394.305	3248.75	358.61	370.928	1083.08	274.27
70	356.000	2502.21	524.31	334.448	950.40	419.15
80	307.459	996.35	610.03	297.321	464.02	529.33
90	276.699	478.62	734.54	N/A	N/A	N/A
Average	**597.197**	**2967.99**	**354.80**	**570.256**	**764.92**	**191.52**

Table 7.13: Results for the conditional $p-$centre problem for $pr439$ where $q = 10$ & 20

Table 7.13 shows that the *CON ERRA* performs better than the *CON CCA* as, on average, optimal solutions were obtained using approximately 81.5% less computational time due to the enhancements previously discussed. Therefore, the *CON ERRA* is used to solve the conditional $p-$centre problem for the large TSP-Library data sets.

Computational Results for the Larger Data Sets

The efficiency of the *CON ERRA* is demonstrated by solving the $(p, q)-$centre problem optimally for the TSP-Library data sets $rat575$, $rat783$, $pr1002$ and $rl1323$ where $q = 20$. The results are produced, for the first time in this literature, in Table 7.14. The first column represents the number of new facilities added, p, and the second and third columns shows the optimal solution value, Z^*, and corresponding computational time in seconds for data sets $rat575$, $rat783$, $pr1002$ and $rl1323$ respectively.

Table 7.14 shows that optimal solutions were obtained for all data sets in a reasonable

amount of time which is an encouraging result. It is important to note that memory issues meant that only an upper bound on the optimal solution for the largest TSP-Library data set $rl1323$ where $p = 70$ was found. In this instance, the best upper bound found (alongside the corresponding computational time) is given. However, all optimal solutions are found within a maximum computational time of 23159.05 seconds for the remaining TSP-Library data sets which is very promising.

	$rat575$		$rat783$		$pr1002$		$rl1323$	
p	Z^*	CPU (secs)	Z^*	CPU (secs)	Z^*	CPU (secs)	Z^*	CPU (secs)
10	45.113	1.59	52.660	2.45	1595.712	1.34	1860.859	2.15
20	39.665	16.62	47.184	18.42	1362.369	40.06	1644.258	35.56
30	32.104	97.47	37.165	182.57	1147.007	264.17	1352.777	603.99
40	26.673	287.22	32.466	1210.18	957.209	1233.67	1111.14	2995.96
50	23.345	627.22	28.306	3010.33	845.932	3561.78	973.382	7971.74
60	21.412	1114.18	25.775	6441.97	755.811	4526.57	857.13	9866.43
70	19.559	1903.97	23.505	13975.64	682.825	6454.95	1137.754^	2391.66
80	17.901	2131.41	21.552	14764.33	622.268	6460.20	718.441	23159.05
Average	**28.222**	**772.458**	**33.577**	**4950.736**	**996.142**	**2817.84**	− − −	**5878.32**

^ Best upper bound obtained (with corresponding time) before memory issues

Table 7.14: Results for the conditional $p-$centre problem where $q = 20$

7.4 Summary

This chapter has adapted $ERRA$ developed in Chapter 5 to optimally solve two related practical location problems. Firstly, it was adapted such that it can be used to solve the $\alpha-$neighbour $p-$centre problem. This was compared to Chen & Chen's adapted classic relaxation algorithm for the TSP-Library data set $pr439$. The Adapted Enhanced Reverse Relaxation Algorithm ($AERRA$) was found to be superior as it required substantially less computational time, demand points in the subset and iterations to solve the problem optimally. This therefore demonstrates that $AERRA$ is the most efficient optimal algorithm known to date to solve this problem type. Optimal results for larger data sets from the TSP-Library are produced for the first time. Furthermore, a scenario analysis was conducted for managerial insight into the case where the coverage need for each demand point varied. The computational results found were very encouraging, as optimal solutions could be attained quickly for all the large TSP-Library data sets.

Secondly, $ERRA$ was incorporated into a known algorithm for the conditional $p-$centre

problem, first proposed by Drezner (1989) and developed by Chen & Chen (2010), to create a more efficient algorithm, *CON ERRA*. Furthermore, two enhancements inspired by *ERRA* were also briefly discussed and added. The computational results showed that the *CON ERRA* was more efficient compared to Chen & Chen's algorithm, and it was further used to optimally solve the conditional $p-$centre problem for the larger TSP-Library data sets for the first time.

The next chapter will summarise the findings of this thesis and outline any areas where further research may be worthwhile.

Chapter 8

Conclusions and Suggestions

This aim of this chapter is to summarise the research and findings given in this thesis and to highlight any research that could be implemented in the future based on the algorithms and ideas developed in this dissertation.

8.1 Conclusion

This thesis has investigated and developed several methods to solve the continuous $p-$centre problem, focusing particularly on optimal methods. The first chapter began by establishing the research question to be thoroughly investigated and answered in this thesis. It then proceeded to provide a brief history on location problems to establish the evolution of research in this area, before analysing the relevant and interesting classifications of the $p-$centre problem. These included classifications such as distinguishing between discrete and continuous problems, identifying the objective function and establishing if the problem is orientated, conditional or constrained. Finally, the exact and heuristic methodologies used to solve location problems were discussed, and the specific methods used in this research were highlighted.

The second chapter provided a literature review on location problems with a focus on the continuous $p-$centre problem. The first section focused on the work directly used in this research, such as the exact methods like Drezner's method, the Elzinga-Hearn algorithm or the relaxation method, as well as the heuristic methods that are also incorporated into this research, such as Cooper's multi start method and Drezner's H_2 heuristic. The second section dealt with the related literature where the ideas and methods were adopted or used as inspiration for this research. This includes the literature for the discrete $p-$centre problem and metaheuristic methods, as well as several variations for the continuous $p-$centre problem such as the constrained $p-$centre problem. The chapter built a clear picture of the development of the continuous $p-$centre problem, critiques the various methods used to solve the $p-$centre problem and high-

lights the areas where further contributions can be achieved.

The third chapter introduced the main formulations, relevant ideas and new methods that were used in the main body of the research. The first section formulated the locations problems that were relevant or directly used in this report, namely the set covering problem, the discrete $p-$centre problem and the continuous $p-$centre problem. For clarity, the objective function and constraints for each problem were clearly indicated and explained. The second section described the preliminary experiments that were incorporated into or used as a springboard for the main body of research conducted in this thesis. This covered an initial research into heuristic methods, such as investigating the embedment of a good quality heuristic solution into exact methods to solve the $p-$centre problem optimally. Furthermore, a new and interesting idea that obtains all the potential facility locations defined by the circles made from three demand points in significantly less computational time compared to a well-known and popular method is described.

The fourth chapter begins the main research completed in this report by examining an interesting optimal algorithm first proposed by Drezner over 30 years ago. The method uses a subset of potential facility locations, namely maximal circles, to optimally solve the $p-$centre problem. Drezner's original algorithm was tested, and this revealed two areas for possible enhancement. The first area enhanced the way maximal circles are identified from one iteration to the next. The second proposed an adaptive policy that finds a compromise solution between the feasible and the optimal solution at each iteration to create an efficient enhanced algorithm for any data distribution type. The Drezner enhanced algorithm (DEA) showed its superiority by producing optimal solutions for the first time for the TSP-Library data sets $rat575$, $rat783$, $pr1002$, and $rl1323$ whilst requiring a 96% decrease in computational time in some instances. It was also tested on generated data sets where $n = 400, 600$ & 800 to show the algorithm's efficiency with different distributions. This illustrated further its effectiveness at optimally solving different distribution types.

The fifth chapter investigated an optimal method used to solve the $p-$centre problem called the relaxation method. This is where large problems are broken down into

smaller sub-problems that are solved successively until optimality is reached. Two new algorithms proposed by Chen & Chen (2009), called binary relaxation and reverse relaxation, are initially investigated. These early investigations revealed that the reverse relaxation showed the most potential for enhancement, and was therefore chosen for further development. Four enhancements were mathematically supported and explained, and incorporated into the algorithm to form the Enhanced Reverse Relaxation Algorithm ($ERRA$). The $ERRA$ was tested against Chen & Chen's original reverse relaxation algorithm, and the results were encouraging as it showed an improved performance compared to Chen & Chen's original reverse relaxation algorithm by requiring 87.64% less computational time to optimally solve the problem. Furthermore, the $ERRA$ is robust as it is deterministic which means that the results can be replicated easily. The $ERRA$ was tested on the TSP-Library data sets mentioned above, and optimal results were obtained for the first time. For consistency, it was also tested with the generated data sets from the previous chapter to show the algorithm's efficiency with different distributions.

The sixth chapter explored a new approach of heuristically solving the $p-$centre problem using relaxation of facilities rather than demand points. The chapter developed a new matheuristic that was used to find a tight upper bound on the optimal solution to the $p-$centre problem. This solution value was then embedded into one of the two optimal methods previously developed in Chapters 4 and 5 to find the optimal solution. The matheuristic incorporates both the well known locate-allocate heuristic method and the enhanced reverse relaxation algorithm developed in Chapter 5. The two most important parts of the matheuristic were identified, and several variants were investigated for each area in order to evolve the most efficient matheuristic. These two areas were a) obtaining a good subset of facilities and b) efficiently changing the neighbourhood to avoid becoming stuck in a local minimum. The best variant in each case was highlighted, and the finalised matheuristic was used to find a tight upper bound on the optimal solution for the TSP-Library data sets. In some instances, the matheuristic found a better solution value than the best known heuristic solution value. For consistency, the matheuristic was also tested on the generated data sets previously mentioned to cast further light on the matheuristic's efficiency with different distributions. The matheuristic was found to be most efficient with clustered data. Interestingly, it was

also found to be most efficient with larger random data sets than smaller random data sets.

The seventh chapter adapted the $ERRA$ to solve the $\alpha-$neighbour $p-$centre problem, whilst also modifying an existing algorithm for the conditional $p-$centre problem to demonstrate the flexibility of this optimal method for location problems related to the classical $p-$centre problem. When studying the $\alpha-$neighbour $p-$centre problem, the Adapted Enhanced Reverse Relaxation Algorithm ($AERRA$) was tested against the adapted classic relaxation algorithm proposed and used by Chen & Chen (2014) to solve the $2-$neighbour $p-$centre problem for the TSP-Library data set $pr439$. The results showed that the $AERRA$ required much less computational time, iterations and demand points in the subset to optimally solve the $2-$neighbour $p-$centre problem. Furthermore, optimal solutions are found for the first time for the data set $pr439$ where $\alpha = 3$. To provide insights for a managerial point of view, a scenario analysis was also conducted where the number of facilities required to cover each demand point varied, and computational results were given for the first time for the TSP-Library data sets that are used throughout this thesis. When studying the conditional $p-$centre problem, $ERRA$ was incorporated into an existing algorithm, and two enhancements were also proposed to create an improved algorithm that showed promising results. This strength of this enhanced algorithm was demonstrated by optimally solving the conditional $p-$centre problem for the large TSP-Library data sets where the number of preexisting facilities was 20.

8.2 Further Research Suggestions

We shall now highlight several areas where further research could be worthwhile exploring in the future.

Firstly, two enhancements proposed in Chapter 4 could be developed further with the aim to create a more efficient algorithm. Firstly, it could be worth investigating if the 'checking area' found using Enh3 could be made tighter with slight alterations to Lemmas 4.4.3 and 4.4.3 in order to speed up the enhanced algorithm further. Furthermore, Enh4 could potentially be made more efficient by first reordering the demand

points in terms of Euclidean distance from each facility, and checking each demand point in order. However, this would involve creating a large matrix of ordered demand points, and so the computational time and computer memory needed to create the matrix must be compared with the overall computational time saved before it can be added to the algorithm as an enhancement.

Secondly, Chapter 5 selected Chen & Chen's reverse relaxation algorithm instead of the binary relaxation algorithm to be enhanced based on its experimental performance. However, had we not tested the two relaxation algorithms ourselves, it would may have been justified to choose the binary relaxation algorithm for further development based purely on Chen & Chen's results. This shows that the binary relaxation algorithm does show potential, and therefore further research and/or similar enhancements for this relaxation-based algorithm may be worthwhile.

Chapter 6 introduced an interesting matheuristic used to find a good feasible solution for the continuous $p-$centre problem. As this is a new and alternative approach to the relaxation method, there are many variants and areas of further research that could be considered. For example, the choice of facilities forming the subset could be explored further. Our algorithm initially chooses only the facilities that form the Voronoi polygon or the neighbouring facilities polygon. However, a second level of each polygon could also be considered for $FSub$. In other words, the Voronoi or neighbouring facilities polygon for each facility in $FSub$ could be found, and the facilities forming these polygons could also be added to $FSub$. This would form a larger $FSub$ initially, but it may yield a more efficient subset and so could be worthwhile exploring.

Finally, Chapter 7 gives two examples that demonstrate how the optimal method developed in Chapter 5 can be adapted for related $p-$centre problems. As these alternate classifications often represent a more realistic location problem, it may be worthwhile adapting the optimal methods given in this report for other classification types. An example of an interesting related problem that could be considered is the previously mentioned constrained $p-$centre problem. The issue here is how to direct the search from an optimal but infeasible solution, towards a feasible one that retains optimal conditions. This is a challenging problem that deserves further investigation.

Bibliography

[1] Albareda-Sambola, M., Dáiz, J., & Fernández, E. (2010). Lagrangean Duals and Exact Solution to the Capacitated $p-$center Problem. *European Journal of Operational Research, 201,* pp. 71-81.

[2] Alharbi, A. M. (2010). Combining Heuristic and Exact Approach for the vertex $p-$centre problem and other related location problems. *PhD Thesis, The University of Kent,* pp. 55-81.

[3] Al-Khedhairi, A., & Salhi, S. (2005). Enhancements to Two Exact Algorithms for Solving the Vertex $P-$Centre Problem. *Journal of Mathematical Modelling and Algorithms, 4, 2,* pp. 129-147.

[4] Averbakh, I., & Berman, O. (1997). Minimax Regret $p-$Center Location on a Network with Demand Uncertainty. *Location Science, 5, 4,* pp. 247-254.

[5] Barber, C., Dobkin, D., & Huhdanpaa, H. (1996). A Quickhull Algorithm for Convex Hulls. *ACM Transactions on Mathematical Software , 22, 4,* pp 469-483.

[6] Berman, O., & Drezner, Z. (2008). A new formulation for the conditional $p-$median and $p-$center problems. *Operations Research Letters, 36,* pp. 481-483.

[7] Berman, O., & Simchi-Levi, D. (1990). The Conditional Location Problem on Networks. *Transportation Science, 24,* pp. 77-78.

[8] Berman, O., Wang, J., Drezner, Z., & Wesolowsky, G. O. (2003). A Probabilistic Minimax Location Problem on the Plane. *Annals of Operations Research, 122,* pp. 59-70.

[9] Brimberg, J., & Mladenović, N. (1996). A Variable Neighbourhood Algorithm for Solving the Continuous Location-Allocation Problem. *Stud Loc Analysis, 10,* pp. 1-12.

[10] Centre of Logistics and Heuristic Optimisation. (2015). The University of Kent. http://www.kent.ac.uk/kbs/research/research-centres/clho/datasets.html (accessed 12/07/2016).

[11] Chen, R. (1983). Solution of Minisum and minimax Location-Allocation Problems with Euclidean Distances. *Naval Research Logistics Quarterly, 30,* pp. 449-459.

[12] Chen, D., & Chen, R. (2009). New Relaxation-based Algorithms for the Optimal Solution of the Continuous and Discrete Problems. *Computers and Operations Research, 36,* pp. 1646-1655.

[13] Chen, D., & Chen, R. (2010). A relaxation based algorithm for solving the conditional $p-$center problem. *Operations Research Letters, 38,* pp. 215-217.

[14] Chen, D., & Chen, R. (2013). Optimal Algorithms for the $\alpha-$Neighbor $P-$Center Problem. *European Journal of Operational Research, 225,* pp. 36-43.

[15] Chen, R., & Handler, G. Y. (1987). Relaxation Method for the Solution of the Minimax Location-Allocation Problem in Euclidean Space. *Naval Research Logistics, 34,* pp. 775-788.

[16] Chen, R., & Handler, G. Y. (1993). The Conditional $P-$Center Problem in the Plane. *Naval Research Logistics, 40,* pp. 117-127.

[17] Church, R. L. (1984). The Planar Maximal Covering Location Problem. *Journal of Regional Science, 24, 2,* pp. 185-201.

[18] Church, R. L., & Gerrard, R. A. (2003). The Multi-Level Location Set Covering Model. *Geographical Analysis, 35, 4,* pp. 277-289.

[19] Church, R., & ReVelle, C. (1974). The maximal covering location problem. *Papers of the Regional Science Association, 32, 1,* pp. 101-118.

[20] Cooper, L. (1963). Location-Allocation Problems. *Operations Research, 3,* pp. 331-343.

[21] Cooper, L. (1964). Heuristic Methods for Location-Allocation Problems. *SIAM Review, 6,* pp. 37-53.

[22] Dantrakul, S., & Likasiri, C. (2012). A Maximal Client Coverage Algorithm for the $p-$center Problem. *Thai Journal of Mathematics, 10, 2,* pp. 423-432.

[23] Daskin, M. S. (1995). *Network and discrete location: Models, algorithms and applications.* John Wiley & Sons.

[24] Davidović, T., Ramljak, D., Šelmić, M., & Teodorović , D. (2011). Bee colony optimization for the $p-$centre problem. *Computers and Operations Research, 38,* pp. 1367-1376.

[25] Davoodi, M., Mohades, A., & Rezaei, J. (2011). Solving the constrained $p-$center problem using heuristic algorithms. *Applied Soft Computing, 11 ,* pp. 3321-3328.

[26] De Berg, M., van Kreveld, M., Overmars,M., & Schwarzkopf, O. (2000). *Computational Geometry.* 2nd Edition, Springer-Verlag, NY.

[27] Drezner, Z. (1984a). The $p-$centre problem Heuristic and Optimal Algorithms. *Operational Research Society, 35,* pp. 741-748.

[28] Drezner, Z. (1984b). The planar two-centre and two-median problems. *Transportation Science, 18,* pp. 351-361.

[29] Drezner, Z. (1987). On the rectangular $p-$centre problem. *Naval Research Logistics, 34*, pp. 229-234.

[30] Drezner, Z. (1989). Conditional $p-$center problems. *Transportation Science, 23,* pp. 51-53.

[31] Drezner, Z. (1995). On the conditional $p-$median problem. *Computers and Operations Research, 22,* pp. 525-530.

[32] Drezner, Z., & Hamacher, H. W. (2001). *Facility Location: Applications and Theory.* Springer.

[33] Drezner, Z., & Shelah, S. (1987). On the Complexity of the Elzinga-Hearn Algorithm for the $1-$Center Problem. *Mathematics of Operations Research, 12, 2,* pp. 255-261.

[34] Drezner, Z., & Wesolowsky, G.O. (1993). Finding the circle or rectangle containing the minimum weight of points. *Location Science, 2,* pp. 83-90.

[35] Dyer, M.E., & Frieze, A.M. (1985). A Simple Heuristic for the $p-$Centre Problem *Operations Research Letters, 3,* pp. 285-288.

[36] Eiselt H. A., & Marianov, V. (2011). *Foundations of Location Analysis.* Springer.

[37] Elshaikh, A., Salhi, S., & Nagy, G. (2015). The continuous $p-$centre problem: An investigation into variable neighbourhood search with memory, *European Journal of Operational Research, 241,* pp. 606-621

[38] Elshaikh, A., Salhi, S., Brimberg, J., Mladenović, N., Callaghan, B., & Nagy, G. (2016). Adaptive Perturbation-Based Heuristics: An Application for the Continuous $p-$Centre Problem. *Computers & Operations Research, 75,* pp. 1-11.

[39] Elzinga, J., & Hearn, D. (1972). Geometric Solutions for some Minimax Location Problems. *Transportation Science, 6,* pp. 379-394.

[40] Gabriel, K. R., & Sokal, R. R. (1969). A New Statistical Approach to Geographic Variation Analysis. *Systematic Biology, 18, 3,* pp. 259-278.

[41] Gomory, R. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society, 64,* pp. 275-278.

[42] Hakimi, S. L. (1964). Optimum Locations of Switching Centers and the Absolute Centers and medians of a Graph. *Operations Research, 12,* pp. 450-459.

[43] Hakimi, S. L. (1965). Optimum Location of Switching Centers in a Communications Network and some related Graphical Theoretic Problems. *Operations Research, 13,* pp. 462-475.

[44] Handler, G.Y. (1978). Finding Two-Centers of a Tree: The Continuous Case. *Transportation Science, 12, 2,* pp. 93-106.

[45] Handler, G. Y., & Mirchandani, P. B. (1979). *Locations on Networks: Theory and Algorithms.* Cambridge, MA: MIT Press.

[46] Hansen, P., Mladenović, N., & Pérez, J. A.M. (2010). Variable Neighbourhood Search: Methods and Applications. *Annals of Operations Research, 175,* pp. 367-407.

[47] Hogan, K., & ReVelle, C. (1986). Concepts and Applications of Backup Coverage. *Management Science, 32, 11,* pp. 1434-1444.

[48] Ilhan, T., & Pinar, M. C. (2001). An Efficient Exact Algorithm for the Vertex $p-$center problem. http://www.optimization-online.org/DB_HTML/2001/09/376.html. (accessed 19/05/14).

[49] Irawan, C. (2014). Investigating Large Conditional and Unconditional $p-$median and $p-$centre problems. *PhD thesis, The University of Kent.*

[50] Irawan, C. A., Salhi, S., & Drezner, Z. (2016). Hybrid Meta-heuristics with VNS and Exact Methods: Application to Large Unconditional and Conditional Vertex $p-$Centre Problems. *Journal of Heuristics, 22* pp. 507-537.

[51] Kavah, A., & Nasr, H. (2011). Solving the conditional and unconditional $p-$centre problem with modified harmony search: A real case study. *Scientia Iranica, 4,* pp. 867-877.

[52] Kennedy, J., & Eberhart, R. (1995). Particle Swarm Optimization. *Proceedings of IEEE International Conference on Neutral Networks,* pp. 1942-1948.

[53] Khuller, S., Pless, R., & Sussmann, Y. J. (2000). Fault tolerant $K-$center problems. *Theoretical Computer Science, 242,* pp. 237-245.

[54] Krumke, S.O. (1995). On a generalization of the $p-$centre problem. *Information Processing Letters, 56,* pp. 67-71.

[55] Kuby, M. J. (1987). Programming Models for Facility Dispersion: The $p-$dispersion and Maxisum Dispersion Problems. *Geographical Analysis, 19,4,* pp. 315-329.

[56] Love, R.F., Morris, J. G., & Wesolowsky, G. O. (1988). *Facilities Location: Models and Methods.* North Holland, NY.

[57] Lowe, T. J., & Wendall, R. E. (2011). Exact Solution of Two Location Problems via Branch-and-Bound. In: Eiselt H. A., & Marianov, V. (Eds). *Foundations of Location Analysis.* Springer, pp. 291-314.

[58] Lu, C. (2013). Robust weighted vertex $p-$center model considering uncertain data: An application to emergency management. *European Journal of Operational Research, 230,* pp. 113-121.

[59] Megiddo, N., & Supowit, K.J. (1984). On the Complexity of some common Geometric Location Problems. *Society for Industrial and Applied Mathematics, 13, 1,* pp. 182-196.

[60] Mehrez, A., & Stulman, A. (1982). The Maximal Covering Location Problem with Facility Placement on the Entire Plan. *Journal of Regional Science, 22, 3,* pp. 361-365.

[61] Meihle, W. (1958). Link-Length Minimization in Networks. *Operations Research, 6, 2,* pp. 232-243.

[62] Minieka, E. (1970). The $m-$centre problem. *SIAM Review, 12,* pp. 138-139.

[63] Minieka, E. (1980). Conditional Centers and Medians on a Graph. *Networks, 10,* pp. 265-272.

[64] Mladenović, N., & Hansen, P. (1997). Variable neighbourhood search. *Computers and Operations Research, 24,* pp. 1097-1100.

[65] Mladenović, N., Labbé, M., & Hansen, P. (2003). Solving the $p-$Center Problem with Tabu Search and Variable Neighborhood Search. *Networks, 42, 1,* pp. 48-64.

[66] Murray, A., O'Kelly, M. E., & Church, R. (2006). Regional Service Coverage Modelling. *Computers and Operations Research, 35,* pp. 339-355.

[67] Murray, A. T., & Wei, R. (2013). A computational approach for eliminating error in the solution of the location set covering problem. *European Journal of Operational Research, 224,* pp. 52-64.

[68] Özsoy, F. A., & Pinar, M. . (2006). An Exact Algorithm for the Capacitated Vertex $p-$Center Problem. *Computers and Operations Research, 33,* pp. 1420-1436.

[69] Pacheco, J. A., & Casado, S. (2004). Solving two location models with few facilities by using a hybrid heuristic: a real health resources case. *Computers and Operations Research, 32,* pp. 3075-3091.

[70] Plastria, F. (2002). Continuous Covering Location Problems. In: Drezner, Z., & Hamacher, H. W. (Eds). *Facility Location: Application and Theory.* Springer, New York, pp. 37-79.

[71] Plesník, J. (1987). A Heuristic for the $p-$center Problem in Graphs. *Discrete Applied Mathematics, 17,* pp. 263-268.

[72] Rabie, H. M., El-Khodary, I. A., & Tharwat, A. A. (2013). A Particle Swarm Optimization Algorithm for the Continuous $p-$center Location Problem with Euclidean Distance. *International Journal of Advanced Computer Science and Applications, 4, 12,* pp. 101-106.

[73] Richard, D., Beguin, H., & Peeters, D. (1990). The location of fire stations in a rural environment: a case study. *Environment and Planning, 22,* pp. 39-52.

[74] Roth, R. (1969). Computer Solutions to Minimum-Cover Problems. *Operations Research, 17*, pp. 455-465.

[75] Salhi, S. (1997). A perturbation heuristic for a class of location problems. *Journal of Operational Research Society, 48,* pp. 1233-1240.

[76] Salhi, S. (1998). Heuristic Search Methods. In: Marcoulides, G. A. (Ed). *Modern Methods for Business Research.* LEA, New Jersey, pp. 147-175.

[77] Salhi, S. (2006). Heuristic Search: The Science of Tomorrow. In: Salhi, S. (Ed). *OR48 Keynotes Papers Operational Research Society*, pp. 39 - 58.

[78] Salhi, S., & Al-Khedhairi, A. (2010). Integrating heuristic information into exact methods: The case of the vertex $p-$centre problem. *Journal of the Operational Research Society, 61,* pp. 1619-1631.

[79] Saski, Y., & Box, P. (2003). Agent-Based Verification on von Thünens Location Theory. *Journal of Artificial Societies and Social Stimulation, 6, 2.* http://jasss.soc.surrey.ac.uk/6/2/9.html (last accessed 21/09/2016).

[80] Schilling, D., Jayaraman, V., & Barkhi, R. (1993). A review of covering problems in facility location. *Locations Science, 1, 1,* pp. 25-55.

[81] Scott, A. J. (1970). Location-Allocation Systems: A Review. *Geographical Analysis, 2,* pp. 95-119.

[82] Shier, D. R. (1977). The Min-Max Theorem for $p-$Center Problems on a Tree. *Transportation Science, 11, 3,* pp. 243-252.

[83] Suzuki, A., & Drezner, Z. (1996). The $p-$Centre Location Problem in an Area. *Location Science, 4,* pp. 69-82.

[84] Suzuki, A., & Okabe, A. (1995). Using Voronoi Diagrams. In: Drezner, Z. (Ed). *Facility Location: A Survey of Applications and Methods.* Springer-Verlag New York, pp. 103-118.

[85] Toregas, C. (1971). Location Under Maximal Travel Time Constraints. *Ph.D. Dissertation, Cornell University.*

[86] Toregas, C., Swain, R., ReVelle, C., & Bergman, L. (1971). The location of emergency service facilities. *Operations Research, 19,* pp. 1363-1373.

[87] Toussaint, G, T. (1980). The relative neighbourhood graph of a finite planar set. *Pattern Recognition, 12, 4*, pp. 261-268.

[88] Travelling Salesman Problem Library. (2015). http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/ (accessed 12/07/2016).

[89] Vijay, J. (1985). An Algorithm for the $p-$center Problem in the Plane. *Transportation Science, 19, 3,* pp. 235-245.

[90] Weber, A. (1909). *Über* den Standort der Industrien, Erster Teil: Reine Theorie des Standortes. T*ü*bingen: Mohr.

[91] Wei, H., Murray A. T., & Xiao, N. (2006). Solving the continuous space $p-$center problem: planning application issues. *IMA Journal of Management Mathematics 17, 4,* pp. 413-425.

[92] Weiszfeld, E. (1937). Sur le point par lequel la somme des distances den points donnés est minimum. *Tôhoku Mathematical Journal, 43,* pp. 355-386.

[93] Welzl, E. (1991). Smallest enclosing disks (balls and ellipsoids). In: Maurer, H. (Ed). *New Results and New Trends in Computer Science.* Springer, pp.359-370.

[94] Wesolowsky, G.O. (1972). Rectangular Distance Location under the Minimax Optimality Criterion. *Transportation Science, 6,* pp. 103-111.

[95] Wesolowsky, G. O. (1977). Probabilistic weights in the one-dimensional facility location problem. *Management Science, 24, 2,* pp. 224-229.

[96] Wesolosky, G. O. (1993). The Weber Problem: History and Perceptives. *Location Science, 1,* pp. 5-23.

[97] Wilamowsky, Y., Epstein, S., & Dickman, B. (1994). How the oldest recorded multiple facility location problem was solved. *Location Science: a multi-disciplinary journal, 3,* pp. 55-60.

Appendix A: Tables

		H_2 Heuristic		Optimal Solution						
n	p	Z	CPU Time	Z^*	Loop CPU Time (secs) [a]	# Loops	Maxi Circles (secs)	CPLEX (secs)	Maxi Circles (%)	CPLEX (%)
439	100	315.486	332.53	256.680	540.75	1428	36.54	464.56	6.76	85.91
575	100	20.402	795.13	20.234$^{\perp}$	57600.00	211	355.60	57238.40	0.62	99.37

[a] This excludes computational time for the H_2 heuristic.
$^{\perp}$Best result found in the time limit of 57600 seconds

Table A.1: Sample results when using when using the $For_0^{(a)}$ formulation with the Drezner enhanced algorithm

	H_2 Heuristic		Optimal Solution						
p	Z	CPU Time	Z^*	Loop CPU Time (secs) [a]	# Loops	Maxi Circles (secs)	CPLEX (secs)	Maxi Circles (%)	CPLEX (%)
90	391.511	276.20	280.90	206.60	393	21.58	159.75	10.45	77.32
100	315.486	332.53	256.68	73.00	200	6.59	57.85	9.03	79.24
Average	**353.50**	**304.37**	**268.79**	**139.80**	**297**	**14.09**	**108.80**	**9.74**	**78.28**

[a] This excludes computational time for the H_2 heuristic.

Table A.2: Sample results for n= 439 TSP-Lib when when using the $For_0^{(b)}$ formulation with the Drezner enhanced algorithm

	H_2 Heuristic		Optimal Solution						
p	Z	CPU Time	Z^*	Loop CPU Time (secs) [a]	# Loops	Maxi Circles (secs)	CPLEX (secs)	Maxi Circles (%)	CPLEX (%)
90	21.932	743.20	17.462	3111.74	2624	165.59	2787.04	5.32	89.57
100	20.402	795.13	16.420	15224.20	1808	94.01	15035.80	0.62	98.76
Average	**21.167**	**769.17**	**16.941**	**9167.99**	**2216**	**129.80**	**8911.41**	**2.97**	**94.16**

[a] This excludes computational time for the H_2 heuristic.

Table A.3: Sample results for $n = 575$ TSP-Lib when using the $For_0^{(b)}$ formulation with the Drezner enhanced algorithm

	H_2 Heuristic	Optimal Solution							
p	Z	Z^*	Loop CPU Time (secs) [a]	# Loops	Max Circles (secs)	CPLEX (secs)		Max Circles (%)	CPLEX (%)
						Total	Last Loop		
10	1716.510	1716.510	347.747	2	283.97	34.62	17.18	81.66	9.95
20	1169.540	1029.710	2880.03	36	362.98	285.11	7.07	12.60	9.90
30	975.000	739.193	2181.77	49	234.27	213.08	3.35	10.74	9.77
40	874.271	580.005	1612.02	84	186.09	241.58	1.84	11.54	14.99
50	580.005	468.542	360.345	177	47.38	207.81	2.06	13.15	57.67
60	570.088	400.195	314.654	177	41.14	165.16	0.71	13.08	52.49
70	503.271	357.946	190.84	160	24.42	110.31	0.55	12.80	57.80
80	467.039	312.500	141.54	165	16.67	86.40	0.36	11.78	61.05
90	391.511	280.903	95.133	165	10.03	67.77	0.32	10.54	71.23
100	315.486	256.680	27.894	65	2.49	20.19	0.38	8.93	72.38
Average	**756.272**	**233.900**	**815.20**	**108**	**120.94**	**143.20**	**3.38**	**18.68**	**41.72**

[a] This excludes computational time for the H_2 heuristic.

Table A.4: Solutions for $n = 439$ TSP-Lib with Enhancements and the Adaptive CPLEX Policy

Appendix B: Contributions

This section outlines the main research contributions. This includes any papers presented at Operational Research conferences, as well as research papers that are accepted, under review or in the preparation stage to be submitted shortly.

Conferences

1. Callaghan, B., Salhi, S., & Nagy, G. (2015). Drezner's Exact Method for the Continuous $p-$Centre Problem Revisited. *EURO Conference 2015,* Glasgow, UK, July 2015. This presentation was based on Chapter 4.

2. Callaghan, B., Salhi, S., & Nagy, G. (2016). Drezners Exact Method for the Continuous $p-$Centre Problem Revisited. *European Chapter on Combinatorial Optimization,* Budapest, Hungary, May 2016 (A refereed conference publication). This presentation was based on Chapter 4.

3. Callaghan, B., Salhi, S., & Nagy, G. (2016). A Powerful Relaxation-Based Algorithm for the $p-$Centre Problem on the Plane. *EURO Conference 2016,* Poznań, Poland, July 2016. This presentation was based on Chapter 5.

4. Callaghan, B., Salhi, S., & Nagy, G. (2016). Enhancing a Relaxation-Based Algorithm for the $p-$Centre Problem.. *International Symposium on Combinatorial Optimisation,* Canterbury, UK, September 2016. This presentation is based on Chapters 5 & 7.

5. Callaghan, B., Salhi, S., & Nagy, G. (2016). Enhancing a Relaxation-Based Algorithm for the $p-$Centre Problem. *OR58 Annual Conference,* Portsmouth, UK, September 2016. This presentation is based on Chapters 5 & 7.

Papers

1. Elshaikh, A., Salhi, S., Brimberg, J., Mladenović, N., Callaghan, B., & Nagy, G. (2016). Adaptive Perturbation-Based Heuristics: An Application for the Continuous $p-$Centre Problem. *Computers & Operations Research, 75,* pp. 1-11. This paper uses results found in Chapter 4.

2. Callaghan, B., Salhi, S., & Nagy, G. Speeding up the Optimal Method of Drezner's

for the $p-$Centre Problem in the Plane. *European Journal of Operational Research, (2016)*, 10.1016/j.ejor.2016.08.038. This paper is based on Chapter 4.

3. Callaghan, B., Salhi, S., & Nagy, G. (2016). A Relaxation-Based Algorithm for the Continuous $p-$Centre Problem. *Omega, (preparation)*. This paper is based on Chapter 5 and Chapter 7.