Adaptive Heuristic Methods for the Continuous *p*-Centre Location Problems

A thesis submitted to Kent Business School at the University of Kent in the subject of Operational Research for degree of Doctor of Philosophy by research and thesis

By

Abdalla Mohamed Elshaikh

Kent Business School

University of Kent

England

May 2014

Synopsis

This research studies the *p*-centre problem in the continuous space. This problem is particularly useful in locating emergency facilities, such as fire-fighting stations, police stations and hospitals where it is aimed to minimise the worst-case response time. This problem can be divided into a single facility minmax location problem (*1*-centre) and multi-facility minmax location problem (*p*-centre). The solution of the *1*-centre location problem can be found optimally in polynomial time by using the well known Elzinga-Hearn algorithm for both the weighted and the unweighted case. The objective of the *p*-centre problem is to locate *p* facilities ($p>1$) so as to minimise the radius of the largest circle. However, in this case, we cannot always guarantee optimality as the problem is known to be NP hard.

The aim of the research is to develop and analyse powerful meta-heuristics including the hybridisation of exact methods and heuristics to solve this global optimisation problem. To our knowledge this is the first study that meta-heuristics are developed for this problem. In addition larger instances previously used in the literature are tested .This is achieved by designing an efficient variable neighbourhood search, adapting a powerful perturbation method and extending a newly developed reformulation local search. Large instances are used to evaluate our approaches with promising results. As all these algorithms use the *1*-centre problem as part of their local search, simple but effective enhancements are designed to speed up the Elzinga-Hearn algorithm.

This thesis is organised as follows:

In chapter 1, a brief review on location analysis and its importance is provided first. This is followed by a description of the problem under study and a motivation for this research including applications and a brief description of a possible classification of location problems. Solution methods with an emphasis on meta-heuristics are used in this study, are also briefly reviewed in this chapter. A detailed literature review on the *p*-centre problem in the continuous space is then given highlighting first the single facility location problem before discussing the multi-facility case. A brief review on exact techniques used for the vertex *p*-centre problem is discussed as this will be integrated into one of our solution approaches when solving the continuous problem.

In chapter 2, we revisit the well-known Elzinga-Hearn algorithm for the *1*-centre location problem in the continuous space for both the unweighted and the weighted cases. Though this algorithm is polynomial, as it is used many many times in our search methods, enhancements that efficiently determine the best points that can be used as initial starting points for the algorithm to cut down some of the unnecessary computations are proposed which displayed interesting results. Furthermore, extra tests using large instances are also used to evaluate the two best enhancements. The best variants are then chosen as part of the local search based on the "locate-allocate procedure" when solving the *p*-centre problem using a simple multi-start approach for illustration purposes. This is then integrated in the other powerful meta-heuristics that are developed in subsequent chapters.

In chapter 3, a Variable Neighbourhood Search heuristic (VNS) which uses two neighbourhood structures namely the customer-based and the facility-based is adapted to solve this related location problem. This is followed by two effective improvement schemes that are introduced in the second phase of our local search based on the well-known Cooper's locate-allocate approach. The idea of covering circles is exploited when defining the neighbourhoods within VNS. A learning mechanism is then incorporated within the search on the best enhancement of the facility-based neighbourhood. Computational results of the proposed enhancements using existing datasets (n=439, 575, 783, 1002 and 1323 TSP-Lib) with various values of p (p=10 to 100 with an increment of 10) demonstrate the power of our VNS-based methods. For the smallest of these instances (n = 439) optimal solutions are reported and used for comparison with the best VNS variant yielding less than half of a one percent.

Chapter 4 provides a brief explanation of the perturbation heuristic originally designed for the *p*-median problem. Two types of perturbations namely a gradual and a strong perturbation are then adapted for our problem where the amount of perturbation is made dynamic. In addition, efficient enhancements are proposed and a learning scheme is embedded into the search. This meta-heuristic shows to be efficient when tested on the same datasets that were used in the previous chapter.

In chapter 5, the idea of using the optimal solution of the discrete problem as the initial solution in the continuous space is given first. The new concept of reformulation local search (RLS) originally designed for the multi-source Weber problem is then adapted and enhanced

further by a scheme for generating a tighter Z value iteratively as the new upper bound when solving optimally the vertex p-centre problem. The idea here is to augment the set of the potential sites by the newly found continuous points at the continuous stage. The shifting between the two spaces continues until no improvement is found. Several stopping criteria are also proposed followed by two enhancements when solving the discrete case including the use of forbidden regions. Finally, extensions of RLS are proposed. These incorporate the use of injection points to allow for diversification of the search and the management of the memory at the discrete phase to control the size of the discrete problem. Computational experiments with encouraging results are reported.

In the last chapter, the conclusions of this research are summarised followed by several suggestions which are believed to be useful for future research.

Acknowledgements

I would like to express my great gratitude to all those contributions which have been essential to the completion of this thesis. First of all, I would like to thank my first supervisor Prof. Said Salhi for his patient supervision and his guidance throughout my research. I am also grateful to my second supervisor Dr Gábor Nagy for his support.

Last but not least, my special thanks go to my beloved wife Fathia and my children, Mohamed, Aisha, Mamoun and Hibah who always encouraged me and understood for not always having me during the weekend. My special thanks also go to my mother who always prays for my success.

I would also like to thank my colleagues Chandra Irawan, Shola, Rahyuda, Dwi, Nurul Huda, Abdullah, and Naveed for their support. Finally, I would like to thank to the Libyan Government and the University of Misurata for the scholarship.

Table of Contents

List of Figures

List of Tables

Chapter 1

Problem Description and Review of the Continuous Centre Problems

A description of the problem under study including its importance is presented followed by some applications and a brief description of a possible classification of location analysis of the *p*-centre problem. The meta-heuristic methods that we aim to investigate in this research are briefly covered. This chapter also presents a literature review on continuous location problems with an emphasis on the *p*-centre problem in the continuous space. Finally, a brief review on the vertex *p*-centre is also provided as some of the methods will be incorporated as part of our methodology when solving the continuous case.

1.1 Facility location problems

Location is one of the most important logistic activities, which is used for the purpose of reducing the costs of the logistics system, improving customer service or reducing delivery of some materials. Location analysis is used to determine the location of facilities in order to contribute to achieving those objectives. An example of a location problem could be a company determining where to locate its plants or warehouses to minimise the distance between these facilities and demand points. It could also be used in the public sector for determining where to locate emergency facilities such as ambulances services, police units and fire stations to minimise the maximum distance or response time (travel time).

1.1.1 The importance of location analysis

The decision maker can use location analysis to answer the following questions:

- How many facilities should be built?
- Where should these facilities be?
- What should the size of the facility be?
- How should demand be allocated?

However, before answering these questions one should determine the purpose of solving the location problem. The difference is mainly whether the problems are in the private sector or in the public sector where the objective function differs from one to the other. For instance, the objective of locating police units may be to minimise the maximum distance between the stations and the demand points (customers). On the other hand, in the case of locating toxic dumps and stations of nuclear power, the location of facilities need to be sited as far as possible from the centres of residential dwelling. Manufacturing companies in the private sector seek to achieve profit maximisation and their aim is to capture the largest possible market share. Therefore, they determine the location of their facilities (plants, warehouses, etc.) on the basis of cost reduction by minimising the total cost of transportation and the fixed cost of establishing such facilities. Note that the location problem does not relate to the siting and size of the facilities only, but also to the allocation of demand points to these facilities, see Eilon *et al*., (1971) and Daskin (1995). For more information on location methods, see Drezner and Hamacher (2001), and Eiselt and Marianov (2011).

1.1.2 Classification of location problem

One of the key criteria used to classify location problems is the topographical criterion, which divides location problems into two basic models, namely continuous and discrete. In continuous location models, there are an infinite number of candidate locations as facilities can be located anywhere on the plane. In discrete location, the sites of the facilities are assumed to occur only on a network or on a graph. In other words, the facilities can be sited only on the nodes or on the links of the network. There is therefore a finite number of candidate locations which are considered as feasible locations of the facilities or also known in the literature as potential sites. Problems can also be divided further into capacitated or uncapacitated, then into single facility or multi-facility. The path that we will follow in this research is shown in bold as represented in Figure 1.1.

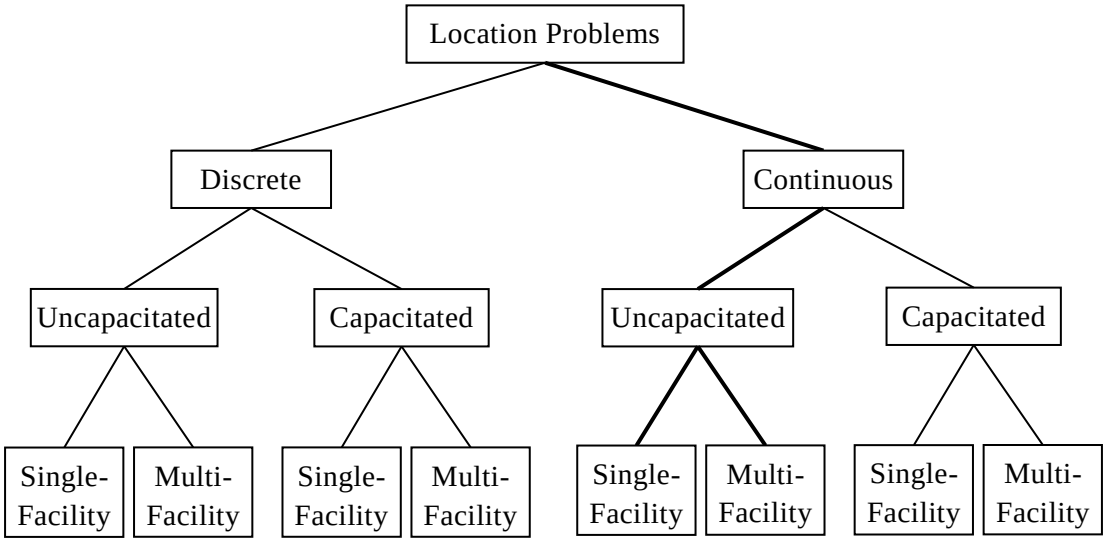


Figure 1.1: Classes of location-allocation problems with bold showing our research path

Facility location problems can also differ in their objective function, which can be grouped into three categories: *p*-median, *p*-centre and covering problems. The next three subsections describe briefly these three models as these will provide research ingredients in our research methodology.

a) p-median problem

The *p*-median problem, originally developed by Hakimi (1964), is to locate *p* facilities with the objective to minimise the demand-weighted total travel distance between the demand points and their corresponding nearest facilities. This is also equivalent to minimising the average weighted distance. Several authors considered that the *p*-median problem is one of the most well-known facility-location problems, which is also called the minisum problem.

b) p-centre problem

The *p*-centre problem was also originally developed by Hakimi (1964, 1965). The objective function is to minimise the maximum distance or response time between the demand points and the nearest facility using a given number (*p*) of facilities. This is also known as the minimax problem. The *p*-centre problem has been used to solve location problems of emergency facilities such as medical facilities (ambulances), police stations,

and fire stations etc where the aim is to minimise the maximum time to respond to an emergency. Other applications include the locations of transmitters to maximise the lowest signal level received, and the location of sirens so to evacuate people as soon as possible, among other applications, Murray and Xiao (2006).

When the location is on the continuous plane, the problem becomes the continuous *p*-centre problem which will be investigated in this research.

c) **Covering problem**

In the covering location model, each demand point is not necessarily assigned to one facility only, but must be allocated to at least one facility. There are two common covering problems, namely the set covering problem and the maximum covering location problem. The first one aims to minimise the number of facilities subject to a prescribed covering distance constraint, whereas the second aims to locate a known number of facilities to maximise the coverage of the demand points (see Daskin, 1995). Here, we briefly describe the former as it will be used in our research. The objective function aims to locate a minimum number of facilities (p) so that all demand points are within a given specified distance or time service (say 4 miles or 10 minutes). The classical model will be revisited as it will be used as part of our search methodology when solving the discrete vertex *p*-centre problem with the aim in generating discrete solutions as initial solutions for the continuous *p*-centre problem.

1.2 The research problem and its impact

This research explores the *p*-centre problem in the continuous space. Here, the facility locations can be located anywhere on the plane. Therefore, their solutions may be infeasible as they may be in a river, a lake or on top of a building. However, such a solution could be used to assist the user to limit the generation of potential sites which can, in some situations, be expensive to obtain. The solution can also be used as a green-field solution as a guide. Research that introduce barriers into the search to avoid such infeasible regions, as investigated by Hamacher and Klamroth (2000) and Klamroth (2001), could also be worth revisiting.

In the continuous space the problem with unweighted Euclidean distances for *n* given points (demand points) has a succinct geometrical interpretation. For instance, when the number of facility locations is one, the problem reduces to find the smallest circle that encloses all the demand points, where the centre (*X*) of this circle is precisely the location of the new facility. This problem is also known in the literature under various names such as the Euclidean *1*-centre problem or the minimum spanning circle. Equivalently, the continuous *p*-centre problem ($p > 1$) seeks to cover the given points (*n*) with *p* circles where the radius of the largest circle is minimised.

For the *1*-centre problem, there are several but similar optimal algorithms including the algorithm of Elzinga-Hearn (1972). Though the Elzinga-Hearn's algorithm is polynomial of the order $O(n^2)$ and hence can be very fast it is still useful to explore this problem as the solution of the *1*-centre problem will be part of the solution of the *p*-center problem when applying and designing heuristics. A short study that aims to enhance its implementation will be carried out in chapter 2. However, the *p*-centre problem is known to be an NP hard problem, see Megiddo and Supowit (1984). In other words, the problem cannot be solved exactly when the number of demand points and facilities are large enough. According to Brimberg *et al*. (2008) in their review paper, there is a rich though limited literature on continuous location problems when compared to discrete location problems. In addition, it is worth noting that the only largest existing data set, where the optimal solutions are known, is the instance with n=439 (TSP-Lib) with p=10 to 100. These results are provided by Chen and Chen (2009). These will be used to evaluate our meta-heuristics. The only other optimal solution method is given by Drezner (1984*b*), but it was limited to n=30 and p=5 only. These two methods will be reviewed later in this chapter. Also, to our knowledge, there are only greedy type methods, and hence this is the first study where meta-heuristics are explored. We examine a variable neighbourhood search, a perturbation-based approach and a reformulation local search. As a by-product in this study good quality solutions for large instances, which have not been done before, will be produced which can also be used as benchmarks for future research.

From a practical view point, it is worth mentioning some real life applications which shows the usefulness of the study. For example, Valinsky (1955) studied a fire-fighting system to determine the optimal location of fire-fighting units in city of New York whereas

Mansfield and Wein (1958) constructed a model to help the management of a railroad, in choosing amongst alternative locations of an automatic classification yard. In 1962, Burstall *et al*. used the Varignon Frame method to determine the location of several factories in London. The optimal location of checking stations on rail lines was explored by Young (1963). For further details, see Drezner and Hamacher, (2001) and recently the edited book by Eiselt and Marianov (2011).

Gleason (1975) used the set covering-based method to find the minimum number of bus stops to guarantee that no customer need walk more than a specified distance to reach a bus stop. This method was also used to locate emergency medical facilities (ReVelle *et al.*, 1977). Mathematical programming models were developed by Saatcioglu (1982) for the airport location selection problem, based on Turkish data. Price and Turcotte (1986) suggested a study that helped the Red Cross find the locations of mobile blood donor clinics in Quebec City, Canada. Other important applications include the locations of broadcasting stations, amplification stations (locations of transmitters) for cellular phones and the locations of the minimum number of defensive missiles, see Ezra *et al*. (1994) and Daskin (2008) for more details. Other applications will be presented in section 1.4 as part of the review section of the *p*-centre problem.

The next section will cover those meta-heuristics that will be adapted in this study.

1.3 Methods relevant to our research

The exact methods are not always practical to solve large combinatorial problems, as these may require an excessive computational time. Therefore, heuristic methods which are approximate methods, have been suggested to find optimal or near optimal solutions within a reasonable amount of computing time. The basic idea of heuristic procedures is that a feasible solution is constructed by using some rules which are based on mathematical logic, common sense and experience. Iterative and improvement procedures are usually used to improve the solution.

The term heuristic is derived from the Greek word *heuriskein*, which means to discover, to find or to explore. There are many definitions of the term heuristic, a modern definition of

heuristics can be found in Reeves (1993) *"*A *heuristic is a technique which seeks good (i.e. near-optimal) solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is".*

As mentioned above, heuristic methods have been used to find near-optimal solutions. The main criteria for evaluating their performance can be classified under two headings; the solution quality and the computational effort, measured in terms of CPU time. Furthermore, other criteria such as simplicity, ease of control, flexibility, interaction and friendliness can also be considered as secondary criteria. For more details, see Salhi (1998).

Based on Salhi (2006), we can classify heuristic methods into three classes, which are: (*a*) only accept improving moves, (*b*) accept improving and non-improving moves and (*c*) use more than one solution at a time. The next subsections will cover the used meta-heuristics in this research that fall under the first category. Here, we describe briefly those heuristics and meta-heuristics that we will explore in this study. These include the classical multi-start approach, variable neighbourhood search, perturbation-based heuristic and reformulation local search.

The classical Multi-Start Approach

The classical Multi-start approach is a randomised approach which starts from a random initial solution at the beginning of each run. The idea is that one of these solutions may lead to the right region which includes the global minimum. However, this approach is considered to be a blind search, as it may revisit the same local minima or poor regions more than once.

In the next chapter, this approach will be used to test the efficiency of our enhancements on the original Elzinga-Hearn algorithm when solving the *p*-centre problem. This is also similar to the heuristic described by Drezner (1984*b*) called H1. In addition, in the implementations of the other meta-heuristics which we explore in this research, the CPU time corresponding to 10,000 iterations of this approach will be used as our stopping condition. Besides, this approach will be used to generate initial solutions for our meta-heuristics.

Variable Neighbourhood Search (VNS)

VNS combines the elements of random search with a systematic way of exploring different regions of the solution space through the use of different neighbourhoods followed by a local search. VNS was first developed by Brimberg and Mladenovic (1996) and Hansen and Mladenovic (1997) to solve the multi-source Weber problem and the *p*-median problem respectively. VNS was also shown to be applied successfully in solving several combinatorial problems (see Hansen and Mladenovic (2003)). The idea of VNS is that a random neighbour of the current solution is generated in the first defined neighbourhood and a local search is performed on it. If a better solution is found, it is chosen and the search reverts back to the first neighbourhood, otherwise the search is enhanced using the next usually larger neighbourhood. This continues until all the specified neighbourhoods have been searched where the algorithm either restarts from the first neighbourhood or stops depending on the allocated CPU. The motivation of VNS is that it provides a systematic mechanism for expanding the neighbourhood search in the area of the solution surrounding a local minimum trap. In other words, VNS is very simple to use and has proved to be very popular as shown by the regular biannual VNS mini conferences (the last one being VNS 2014 having taken place in Djerba, Tunisia this October) and the special issues dedicated to the area (the last one being Computers and Operations Research edited by Mladenović *et al*. (2014)). In this research, we will explore the power of VNS while introducing new ingredients into the search. This will be investigated in chapter 3.

Perturbation-Based Heuristic

In this approach some perturbations or noises are introduced to explore infeasible solutions while guiding the search. This approach was adopted by Salhi (1997) for the *p*-median problem as well as for the uncapacitated facility location problem. The idea is to allow the number of facilities of a solution to go over and under the required number of facilities (p), by a certain amount (q). Hanafi and Freville (1998) and Zainuddin and Salhi (2007) also adapted this perturbation-based heuristic for the knapsack problem and the capacitated multi-source Weber problem respectively. In this research we will revisit this approach in chapter 4 while considering the characteristics of the continuous *p*-centre problem.

Reformulation Local Search (RLS)

Very recently, Brimberg *et al*. (2014) developed a new local search for solving the multi-source Weber problem. In the reformulation local search (RLS), the discrete location problem is first solved optimally or heuristically by using the demand points as potential sites. This is then followed by applying a local search to turn such a solution into a continuous one (local optimum in the continuous space). The new obtained continuous locations are then added as potential sites to the problem. The augmented discrete location problem is then solved and its new solution is used again as a starting point for the continuous problem. This procedure, which shifts between the two spaces, continues until a stopping condition is met usually when no improvement can be found in solving the augmented discrete problem. It is worth noting that the idea of using the discrete solution as a starting solution for the continuous space, but without adding the new continuous locations to the problem, was originally developed by Hansen and Mladenović (1998) and also used by Gamal and Salhi (2001) for the multi-source Weber problem where an exact method is used in the former and a heuristic in the latter when solving the discrete problem. This new RLS approach, which to our knowledge was used only once, will be revisited and extended in chapter 5.

1.4 Continuous centre problems-a review

1.4.1 Introduction

The objective of the *p*-centre problem is to minimise the maximum distance between a demand point and its nearest facility. The *p*-centre problem also fits in the class of minimax problems. For a review on continuous covering problems including a history of the *p*-centre problem, see the comprehensive chapter by Drezner (2011) given in the edited book by Eiselt and Marianov (2011) which also includes various location methods and applications. For more references see the edited book by Drezner and Hamacher (2001).

The next two sections cover the single and the multiple facilities cases respectively. These are reviewed here as they make up the basis of our research.

1.4.2 Single facility minmax location problems (*1*-centre)

According to Eiselt and Marianov (2011), the *1*-centre location problem with unweighted Euclidean distances has a long history. In 1857, the English mathematician James Joseph Sylvester (1814-1897) discussed this problem by posing the question of finding the smallest circle that encloses a given set of points in the plane (Sylvester 1857). In 1860, he published the analysis of this problem and other related problems and he noted that the optimal solution can be determined by two or three points. The solution of the latter is based on whether the triangle is obtuse or acute. Sylvester then continued to describe his algorithm based on a solution method of Peirce with no citation. For more details, see Eiselt and Marianov (2011, p. 63-72).

According to Hurtado *et al*., (2000) Sylvester's algorithm is very simple and is based on the geometric fact that the smallest circle is determined by either two or three points. The method for obtaining a solution is briefly described here: (i) for every pair of points determine their midpoint and its corresponding circle and for every three points determine also the centre of the circle, (ii) for every circle check if all points are encompassed by it and (iii) out of all such "feasible" circles choose the smallest one. His crude algorithm, guarantees optimality and has a time complexity of $O(n^4)$, where *n* is the number of demand points.

In 1885, Chrystal unknowingly reinvented the approach of Sylvester. Chrystal observed that the circle that includes all the vertices of the convex hull does also include all the set of *n* points. Therefore, he excluded all the points that are not vertices of the convex hull from the problem, which may significantly decrease the number of the points which determines the problem. The algorithm of Chrystal starts with a large circle that includes all the demand points and decreases the radius of the circle iteratively, until the optimal solution (smallest circle) is reached. In other words, the radius is reduced by examining a pair of vertices of the convex hull with the other demand point at each iteration. Chrystal showed that the bound on the number of iterations is $\frac{1}{2}m(m-1)$, where *m* refers to the number of vertices in the convex hull ($m < n$). This is because there are $m(m-1)/2$ possible segments connecting the vertices and any segment cannot be considered more than once.

The unweighted *1*-centre problem in the continuous space can also be formulated as follows:

Given n distinct points $P_i = (a_i, b_i)$; $i = 1, \ldots, n$ in the plane, find a point $X = (x, y) \in \Re^2$ that minimises the maximum Euclidean distance from X to the given points $\left(\text{i.e., } d(X, P_i)\right)$

Let $f(X) = \max_{1 \le i \le n} d(X, P_i)$. The problem is to minimize $f(X)$, *i.e.*

$$\min_{X \in \Re^2} \max_{1 \le i \le n} d(X, P_i).$$

A standard transformation is to write the problem as follows:

$$\min Z$$

$$\text{s.t. } d(X, P_i) \le Z \text{ for } i = 1, \ldots, n.$$
$$X \in \Re^2$$

Bass and Schubert (1967) improved the performance of Sylvester's algorithm (1857) by determining the set of extreme points of a given set of points of finite cardinality in the plane. Firstly, an initial set of extreme points is chosen on the basis of extreme values of the coordinates. These are the two points with the largest and the smallest X-axis and the two points with the largest and the smallest Y-axis. Secondly, finding the extreme points of the convex hull, the algorithm of Sylvester is then used. This algorithm runs in $O(h^4 + n \, Log \, n)$ time, where h is the number of extreme points of the convex hull. In the next chapter, the idea of choosing the two points with the largest and the smallest X-axis and Y-axis will also be used to enhance the algorithm of Elzinga-Hearn (1972). Drezner and Shelah (1987) showed that the complexity of the Elzinga-Hearn algorithm is $O(n^2)$.

Eiselt and Marianov (2011) tested the efficiency of the Sylvester-Crystal (SC) algorithm and Elzinga-Hearn (EH) algorithm on instances varying in size from n=10 to 10,000, where they showed EH is more efficient than SC. They also showed that one iteration of SC consumes relatively more CPU time than the EH algorithm which requires fewer iterations than SC, especially when n is large ($n \ge 200$).

Drezner and Wesolowsky (1980) provided an algorithm for the weighted *1*-centre location problem with l_p distances. The algorithm starts by choosing randomly an initial point and the three points whose weighted distances are the farthest from this initial point. The optimal solution *Z* (*1*-centre problem when n=3) for these three points is first found. If the weighted distance from the solution to any point is not greater than the value of *Z*, the search stops and the optimal solution is found, otherwise the fourth point that has the greatest weighted distance from the solution point is determined and the three points that have the maximum weighted distance are selected among the four points. The procedure is repeated until the search stops. This is almost identical to the Elzinga-Hearn (1972) algorithm. The authors also provided a formulation to find the optimal solution for the three points (n=3) when using the Euclidian distance. The idea of choosing the three points that have the maximum weighted distances will also be examined in the next chapter.

The work of Carbonet and Mehrez (1980) dealt with the single facility location problem which minimises the maximum rectilinear distance where the locations of prospective demand points are considered to be random variables. Through the concept of the expected value of perfect information, it was shown for the one-dimensional location decisions that a substantial reduction in the maximum distance can be realised by the adoption of a wait-and-see policy.

The work of Chakraborty and Chaudhuri (1981) was a note on a geometrical solution for the single facility minimax location problem with Euclidean distances. The basic idea was similar to the one developed by Peirce (reported by Sylvester (1860)) and by Chrystal (1885). Chrystal used a circle of infinite radius through two adjacent points on the convex hull as the starting solution, while Peirce's algorithm reported by Sylvester used the minimum angle rule instead. The starting point of Chrystal-Peirce algorithm for the unweighted case is to construct a large circle which covers all the points P_i, and which passes through two points, P_s and P_t, and then define X_k as the centre of the circle, and $S_k = \{P_s, P_t\}$. Let $\angle P_s P_v P_t = \min\{\angle P_s P_j P_t: P_j \notin S_k\}$, if $\angle P_s P_v P_t$ is obtuse, stop (the minimum circle has diameter). Otherwise, compute the centre of the circle, passing through P_s, P_t, and P_v. If the triangle $\Delta P_s P_t P_v$ is not obtuse, stop, else, drop the point among P_s, P_t, and P_v with the obtuse angle. Rename the remaining points P_s and P_t, and repeat this procedure by using these points as initial starting points. To a large extent, the efficiency of the Chrystal-Peirce

algorithm depends on how the starting point is chosen. Chakraborty and Chaudhuri used a different starting point and proposed the following by selecting a centre X and let P_s be the farthest point from X. The triangle $\Delta X P_s P_t$ with the hypotenuse along $X P_s$ is constructed and P_t is selected as the point that maximises the hypotenuse ($P_t \underset{P_S \neq P_j}{Max} l(P_S, P_j) / (\overline{P_s P_j} . \overline{P_s X}$)). The centre of the new circle is at the intersection of this hypotenuse with the perpendicular bisector of $P_s P_t$.

Hearn and Vijay (1982) presented a classification scheme for the single facility minmax location problem with respect to both the weighted and the unweighted Euclidean distances. When the positive weights are all equal, this reduces to the equiweighted (unweighted) problem. They also proposed an extension of the Chrystal-Peirce algorithm to the weighted case by proposing how the sequences of the centres of the circles are determined from one iteration to the next.

Megiddo (1983) presented an algorithm for the weighted Euclidean distance which run in $O(n (\log n)^3 (\log \log n)^2)$ time. The author mentioned that for the unweighted case, the most efficient algorithm known is an $O(n \log n)$ algorithm which is proposed by Shamos and Hoey (1975). This algorithm utilises the data structure known as "Farthest point Voronoi diagram". It is not clear whether the generalisation of this concept into the weighted Voronoi diagrams might yield equally efficient algorithms for problems such as the weighted *1*-centre location problem. For the weighted case the problem can be solvable in $O(n^3)$ time on the assumption that these techniques can solve quadratic equations in constant time.

According to Maffioli and Righini (1994), an iterative algorithm for solving the *1*-centre problem on the plane with both the weighted and the unweighted Euclidean distance was presented in Righini (1993). This algorithm is similar to the algorithm given by Hearn and Vijay (1982). The algorithm starts by choosing randomly three points (P_s, P_t and P_u) among the n demand points. Let X be the centre of the circle and find the farthest point from X and call it P_s'. If this point is encompassed by the current circle, stop as the circle covers all points and the solution is optimal. Otherwise find the farthest point from P_s' and call it P_t', compute the midpoint between these two points (P_s' and P_t') and call it M, and then find the farthest point from M and call it P_u'. Finally, rename these three

points (P'_s , P'_t and P'_u) as (P_s , P_t and P_u) and repeat this procedure by using these points as initial starting points. The complexity of this algorithm is $O(n^2)$.

Hurtado *et al.* (2000) studied constrained versions of the Euclidean *1*-centre location problem. The authors provided an $O(n + m)$ time algorithm for the problem of finding the smallest circle that covers a given set of n points with a centre constrained to satisfy m linear constraints. This is an interesting area that could be explored in the future.

1.4.3 Multi-facility minmax location problems (*p*-centre)

The minimax location-allocation problem or the *p*-centre problem as commonly known in location theory, is the following: given n demand points on the plane and a weight associated with each demand point (w_i, i=1,…,n), find p facilities on the plane that minimise the maximum weighted distance between each demand point and its closest new facility.

This problem can also be considered as a MinMaxMin type problem with the following objective function:

$$Z = \underset{X}{Min}\, \underset{i=1,...,n}{Max}\ [w_i\ \underset{j=1,...,p}{Min}\ d(P_i, X_j)]$$

where

n : the number of demand points (fixed points or customers)

p : the number of facilities to open

$P_i = (a_i, b_i)$: the location of demand point i $(i = 1,...., n)$

$w_i > 0$: the weight of demand point i $(i = 1,...., n)$

$X = (X_1,...., X_p)$: the decision variables vector related to these p facility locations with

$X_j = (x_j, y_j)$ representing the location of the new facility j with $X_j \in \Re^2; j = 1,..., p$

$d(P_i, X_j)$: the Euclidean distance between P_i and X_j $(i = 1,...., n; j = 1,...., p)$

In general, the continuous *p*-centre problem has not received much attention from researchers compared to other facility location problems such as the *p*-median problem. It is

worth noting that most of the *p*-centre studies used greedy type heuristic methods with the rest covering a handful of exact methods. To our knowledge, there is not a single meta-heuristic for this continuous problem. Our study will try to fulfil some of these gaps. The review of the solution methods for this problem will be presented under two subsections, namely heuristic and exact methods.

1.4.4 Heuristic methods for the *p*-centre problem

Chen (1983) proposed an algorithm to solve both the minimax and the minisum location-allocation problems with weighted Euclidean distances. This method is based on providing differentiable approximations to the location-allocation objective functions. Thus, if *p* facilities are required to be located with respect to *n* given points, the problem reduces to minimising a nonlinear unconstrained function with the $2p$ variables $x_1, y_1, ..., x_p, y_p$. In both the minisum and the minimax location problems, instances with up to n =100 and p = 10 were solved. Since both the original problems and their approximations are neither convex nor concave, in the large problems the solution was usually only a local minimum, whose solution quality could depend on the initial starting points. The author also discussed the possibility of extending this approach to cater for the case where the costs are not necessarily proportional to the Euclidean distances.

Drezner (1984*b*) presented two heuristics and an optimal algorithm for the *p*-centre location problem with weighted Euclidean distance. The first heuristic (H1), which is of a multi-start type that is similar to the "alternate location-allocation method" proposed by Cooper (1963 and 1964) for the Weber problem except that in the location phase the *1*-centre problem is solved instead of the *1*-median problem. This heuristic starts by randomly choosing *p* points out of the *n* demand points as centres for the problem. Each demand point is then allocated to its nearest centre making *p* distinct partitions where the *1*-centre problem is solved for each partition. If the solution is improved (there is a change in any new location), this process (location-allocation phases) is repeated until there are no changes in the location of the facilities or the allocation of demand points. This algorithm is repeated several times using different starting random locations and the best solution is chosen as the final solution of the problem. The second heuristic (H2) is an enhancement of H1, which incorporates the allocation of the critical points from one partition (circle) to another. The

H1 heuristic will be used in our testing in the next chapter. Also, we adapt H2 in our local searches which will be presented in chapter 3. The optimal algorithm will be explained in the next subsection.

Watson-Gandy (1984) presented an algorithm to solve the problem on the plane with weighted Euclidean distances. The author proposed a suitable approach to generate partitions of demand points by using a given value for the radius of the largest circle R. For any two points (P_s and P_t), if $d(P_s, P_t) > 2R$ or $d(P_s, P_t) > r_s + r_t$ (where $r_s = R / w_s$, $s = 1,\ldots,n$), then these two points are allocated to different facilities, otherwise the value of R is exceeded. The author also used the graph colouring algorithm proposed by Brown in (1972) to construct these partitions. This algorithm was examined on instances from $n = 10$ to 50 demand points with differing weight ranges.

Dyer and Frieze (1985) described a simple and fairly intuitive heuristic to solve the weighted p-centre problem in the plane. They chose a point of the largest weight for the first centre, and then they successively selected new centres so that the next centre selected at the point that has the largest weighted distance from its nearest centre. This scheme was repeated until the required number of centres is achieved. This has the useful property that the solution of $(p+1)-$centre is a superset of the p-centre solution. However, this scheme can also be seen to be restrictive as the optimal solution is unlikely to have such a property. Their heuristic requires only $O(np)$ distance evaluations, arithmetic operations and comparisons. The authors also showed that the ratio of the objective function value of their heuristic solution to that of the optimum is bounded by min $(3, 1+\alpha)$ with $\alpha = \frac{Max\ w_i}{Min\ w_i}$.

Eiselt and Charlesworth (1986) designed three constructive heuristic methods known as SWITCHOFF, CRITICAL and STEPDOWN to solve the p-centre problem in the plane with the unweighted Euclidean distance. In all these algorithms, the Elzinga-Hearn algorithm (1972) was used to solve the *1*-centre problem within each cluster. The first method closely resembles the "alternate location-allocation method" proposed by Cooper (1963 and 1964). The only difference is due to the different objective functions, where the authors used the Elzinga-Hearn algorithm instead of the Weiszfeld algorithm which is

used to solve the *1*-median problem. The second algorithm is related to the "vertex substitution method" of Teitz and Bart (1968) which was suggested to solve the *p*-median problem. The main idea is to reallocate one of the critical points of the largest circle (points on its circumference) to another circle (non-critical clusters). If the solution is improved (reduction in the critical distance), the new solution is recorded and the procedure is repeated from this new solution; otherwise we retain the old solution and reassign another critical point. Note that this method is similar to the H2 heuristic given by Drezner (1984*b*). The third method (STEPDOWN) is a technique, which starts with *n* open centres by choosing all the *n* demand points. In each iteration, the two clusters with the closest facilities are combined and their new centre is found. This reduction scheme is repeated until the required number of centres say *p*, is reached. In chapter 3, the idea of this second algorithm (CRITICAL) will be adapted in order to be used to improve the performance of our local search.

Ezra *et al.* (1994) developed an algorithm for the location of the *p*-centre problem with Euclidean distances. This is based on the repeated solution of finite relaxation problems using an interactive computer graphical method to find the locations of *p* circles that had very similar radii to cover all demand points. The user needs to find the initial points to be included in the relaxation set and to inspect on the screen at every stage whether the displayed solution, as demonstrated by circles covering the demand region, is feasible. If it is not, a new demand point is selected and added to the relaxation set. Here, the authors extended the method of Chen and Handler (1987) of optimally solving the *p*-centre location problem which we will explain next in the exact methods subsection.

A Voronoi Diagram Heuristic (VDH) was proposed by Suzuki and Okabe (1995) to solve the continuous *p*-centre problem for area coverage instead of point coverage. This is applicable for instance in agriculture, fire forest protection, irrigation and warning sirens, etc. The idea of VDH for the continuous *p*-centre problem is to choose randomly *p* points (*p* centres) on the plane as an initial configuration. The Voronoi diagram is then constructed using these *p* centres, to form a set of polygons $V = \{V_1, \ldots, V_p\}$ called 'Voronoi polygons'. The centre of each Voronoi polygon is then computed by solving the *1*-centre problem with respect to the vertices of the polygon. This process continues until the centre locations do not change or the required number of iterations (maximum number of iterations) is achieved.

Suzuki and Drezner (1996) investigated the *p*-centre problem for demand originating in an area rather for the special case of a square area. The authors suggested a heuristic solution procedure based on the VDH of Suzuki and Okabe, followed by a finishing up algorithm based on a non-linear programming formulation to improve the solution further.

Pelegrin and Canovas (1998) proposed a new assignment rule (NAR) based on seed points for the continuous *p*-centre problem. The authors suggested three modifications to generate a set of *p* seed points, $S = \{S_1,...,S_p\}$. The first modification extends the method given by Dyer and Frieze (1985). Here, the two farthest points are chosen as the first two seed points (S_1 and S_2) instead of choosing them randomly any point (S_1). The choice of the other seed points is the same for both the other two procedures, where the new seed point is selected as the farthest demand point from the seed points previously generated. In the second modification, the authors suggested using the first seed point as the farthest demand point from the gravity centre of the demand points. The third modification extends the algorithm provided by Plesnik (1987) which was proposed for the *p*-centre problem in graphs. In the allocation part of the heuristics, each demand point is usually allocated to its nearest facility, but in the original class of seed points algorithms, every point is allocated to its nearest seed point. It was shown, using computational experiments, that both the running times and the solutions quality are significantly improved by this new assignment rule.

Wei *et al*. (2006) proposed the constrained Voronoi diagram heuristic (CVDH) for solving the *p*-centre location problem for area coverage by extending the method (VDH) given by Suzuki and Okabe (1995). The main difference between VDH and CVDH is that in the first step of the original VDH *p* centres are generated randomly in the region as an initial configuration, while in CVDH *p* centres are generated randomly in the feasible siting region which may not be convex. The authors also adapted constrained minimum covering circle to solve the *1*-centre problem by modifying the way the new *1*-centre problem was solved by controlling feasibility. The idea was also initially discussed by Plastria (2002). A case study for emergency warning sirens in Dublin, Ohio was used to demonstrate the usefulness of this practical modification.

To our knowledge, there is no meta-heuristic for this problem. We aim to remedy this weakness by addressing this issue through the design and analysis of three types of meta-heuristics that will be given in the next chapters.

1.4.5 Exact methods for the *p*-centre problem

Drezner (1984*a*) proposed two optimal algorithms for the solution of the *2*-centre and the *2*-median location problems with Euclidean distances on the plane. The idea is that any two demand points can be separated by a straight line. Since the optimal location of a facility within every set can be found optimally, the problem reduces to finding an efficient way of defining these straight lines. The author efficiently solved problems up to 100 demand points. However, for the minimax version, he just applied equal weights (unweighted case), and solved the single-facility problem using the method of Drezner and Wesolowsky (1980).

As mentioned earlier, Drezner (1984*b*) also presented a polynomial algorithm for solving the *p*-centre location problem optimally. The main steps of this algorithm are the following: Find a feasible solution to the problem using any heuristic and let Z_0 denote the value of the objective function as a feasible solution. Construct all maximum sets based on Z_0 and find a feasible solution for the corresponding a set covering problem. A maximum set with respect to Z_0 is the set of demand points encompassed by a circle of (*i*) a radius $< Z_0$ and (*ii*) any additional external point if added will make the radius of the new circle $> Z_0$. If a solution is found, use the new Z as Z_0 and repeat, otherwise Z_0 is the optimal solution. The optimal algorithm was tested on small instances varying in size from n=10 to 60 solved with p= 2, from n=10 to 50 with p= 3, from n=10 to 40 with p= 4 and from n=10 to 30 with p= 5. It could be interesting to explore this optimal approach further though no one has pursued it so far. Note that the construction of the maximum set can be time consuming especially if the process is repeated several times.

A similar algorithm was proposed by Vijay (1985) where the sequences of *p*-cover problems assuming a given radius were solved by a zero-one integer programming code. Some simple geometric properties were also used to generate each *p*-cover problem much more efficiently.

Chen and Handler (1987) adapted the relaxation method originally suggested for the *p*-center problem on a network by Handler and Mirchandani (1979), to the *p*-centre problem in the continuous space with Euclidean distances. For any optimal solution, the locations of these circles will be among the finite number of circles $n + \binom{n}{2} + \binom{n}{3}$, where n is the number of null circles (a service point at a demand point), $\binom{n}{2}$ is the number of circles determined by two points on the two ends of a diameter and $\binom{n}{3}$ is the number of circles determined by three points on the circumference (edges of an acute triangle). It is worth noting that the number of these circles becomes very large when *n* is large. In this work, the authors introduced useful ideas to reduce this total number. The main idea is that a relaxed problem using a subset of *m* points out of the *n* points ($m << n$) is chosen and the optimal solution is found using a set-covering algorithm to find a set of *p* circles which cover all the *m* points in the relaxed problem. If the solution is feasible for the original problem, then it is also optimal for the original problem. Otherwise, the relaxed problem is augmented by adding a point and the procedure is repeated. Here the authors suggested adding the farthest point from its nearest centre. It is worth noting that at each iteration any circle with a radius larger than or equal to the current solution (R_{max}) is removed from further consideration which reduces significantly the number of circles to be examined in the next set covering problem to be solved.

Chen and Chen (2009) presented new variants of a relaxation algorithm for both the continuous and the discrete *p*-centre problems. The relaxation method for the minimax location problem is an algorithm to optimally solve a location problem by solving a succession of small sub-problems which get slightly larger in size with the number of iterations. In creating new relaxation algorithms, the authors were guided by three factors, namely (1) the sizes of the sub-problems, (2) the number of sub-problems and (3) the values of the coverage distances. The authors reported excellent computational results for both the continuous and the discrete *p*-centre location problems, especially for the latter where several large instances are used. The authors only reported the optimal solutions for the $n = 439$ TSP instance using several values of *p* and hence which we will use these

in our testing. For completeness, these new relaxation algorithms are therefore described here.

- ***Enhancements on the classical relaxation algorithm***

In this section, the authors proposed two simple modifications on the classical relaxation algorithms to improve its performance, which are as follows:

i) Efficient updating of the upper bound

The first improvement of the relaxation algorithms is a simple change to the classic relaxation algorithm of Chen and Handler (1987) which led to a considerable effect on its performance. In the previous algorithm, when the feasible solution for the relaxed problem is found, we have to check if it is also feasible for the original problem. Here, the authors mentioned that the *p*-centre problem has two equivalent interpretations. In the first one, we need to locate *p* circles that cover all the points and the aim is to minimise the radius of the largest circle. In the second interpretation, they said that a set of *p* circles is a feasible solution for the problem, if these circles cover all the points. Here, the authors look at the set of *p* circles (at most) that cover all the points in the sub-problem, and then they check whether these also cover all the points in the original problem. In other words, the authors viewed the *p*-centre problem as a problem of locating a set of *p* service points, rather than locating a set of *p* circles, then any set of *p* service points is a feasible solution (not necessarily optimal) for the original problem. Therefore, they considered the feasible solution that consists of the centre of the circles that cover the points in the sub-problem. This is also a feasible solution for the original problem but they check if its value is less than the current upper bound. If it is, the best candidate and upper bound are both updated, and the search continues. This simple change has a very positive effect on the performance of the new relaxation-based algorithm.

ii) Adding more than one point

In the classic relaxation algorithm, a single demand point is added to the sub-problem. However, the authors suggested to add more than one point at a time as it may reduce the number of "uninformative" steps, and thus reduce the number of sub-problems. On the other hand, if too many points are added at once, the smaller sub-problems may become too big to be solved. They also suggested adding the *k* demand points that are farthest from

the service points of the current feasible solution. The choice of *k* can be crucial to the success of their method.

- ***Reverse relaxation***

In the classic relaxation algorithm, they start with an upper bound of infinity, and continue to reduce it until the value of the optimal solution is reached. However, here, they started with a lower bound of zero instead, and constantly increasing it, until the value of the optimal solution is reached. The authors combined relaxation with the approach used for the discrete problem that was suggested by Ilhan *et al.* (2002) and which consists of two phases. In the first phase, a tight lower bound on the optimal solution is computed using LP only and in the second one the lower bound is gradually increased until the value of the optimal solution is reached. Note here that any LP solution which is not feasible is not put forward for the ILP in phase 2 which reduces the number of unnecessary ILP problems to be solved. This algorithm (*Reverse Relaxation*) is based on two facts. The optimal solution of the *p*-centre problem for the relaxed problem is a lower bound on the optimal solution of the original problem. This helps the authors to find tight lower bounds on the optimal solution, and also show that if an optimal solution for a relaxed problem covers all of the points, such a solution is also optimal for the original problem. The second fact shows that we can limit ourselves to a finite set of possible values for the objective function value. This helps to find the optimal solution, by going over all the finite set of possible values for the solution, starting from the current lower bound until a solution is found.

- ***Binary relaxation***

The final relaxation algorithm, known as the binary relaxation, is similar to the one used by Daskin's (1995, 2000) for the vertex *p*-centre problem. Binary relaxation solves relatively few sub-problems with typically small coverage-distance values. At every step in the binary search, we check if there is a solution to the sub-problem with a value less than the *Coverage Distance* or not. If not, their lower bound is updated to be the *Coverage Distance*, otherwise there is a solution to the sub-problem. Here, they need to check if it is a feasible solution to the original problem. If it is, they update the Upper Bound to be the value of the solution, otherwise they add points to the sub-problem. In order to determine the optimal solution and to see whenever they need to update the Upper Bound, they also check if there is a solution to the sub-problem with a value less than the Upper Bound or not. If

the problem is infeasible, then the search stops with the optimal solution being the value of the current Upper Bound.

1.5 Techniques for the vertex *p*-centre problem

The facility locations can be restricted to the nodes of the network; in this case the problem is referred to as a vertex centre problem. The centre problem that allows facilities to be sited anywhere on the network is known as the absolute centre problem. As the discrete *p*-centre problem will be solved as part of our research methodology the two methods that are commonly used in the literature are briefly described here. These include the 0-1 ILP formulation and the set covering-based approach. The first one is used here for completeness whereas the second one will be discussed in this section and revisited in chapter 5 as we will use it in our research.

0-1 ILP formulation

The binary linear programming formulation of the vertex *p*-centre location problem is as follows:

$$\textit{Minimise} \qquad D \tag{1.1}$$

Subject to:

$$\sum_{j=1}^{m} Y_{ij} = 1 \qquad \forall i = 1,...,n, \tag{1.2}$$

$$Y_{ij} \le X_j \qquad \forall i = 1,...,n; \;\; j = 1,...,m \tag{1.3}$$

$$\sum_{j=1}^{m} X_j = p, \tag{1.4}$$

$$D \ge \sum_{j=1}^{m} d_{ij} Y_{ij} \qquad \forall i = 1,...,n, \tag{1.5}$$

$$X_j, \; Y_{ij} \in \{0, 1\} \qquad \forall i = 1,...,n; \;\; j = 1,...,m \tag{1.6}$$

where

I = set of demand nodes, $I = \{P_i, i=1, \ldots, n\}$

J = set of candidate facility sites, $J = \{F_j, j=1, \ldots, m\}$

D = maximum distance or time between a demand node and the nearest facility

p = The number of facilities to be located

$d_{ij} = d(P_i, F_j)$ = the distance between demand node P_i and candidate facility site F_j

$$X_j = \begin{cases} 1 & \text{if a facility is located at candidate site } F_j \in J, \\ 0 & \text{otherwise,} \end{cases}$$

$$Y_{ij} = \begin{cases} 1 & \text{if demand node } P_i \in I \text{ is assigned to an open facility at candidate site } F_j \in J, \\ 0 & \text{otherwise,} \end{cases}$$

The objective function (1.1) aims to minimise the maximum distance between each demand node and its closest open facility. Constraint (1.2) guarantees that each demand node is assigned to exactly one facility. Constraints (1.3) restrict demand nodes to be assigned to open facilities. Constraint (1.4) ensures that p facilities are to be located. Constraints (1.5) guarantee that the maximum distance between a demand node and its nearest facility is greater than or equal to the distance between any demand node and the nearest facility. Finally, constraints (1.6) are the integrality constraints.

For fixed values of p facilities, the vertex p-centre problem can be solved in polynomial time, this can be achieved by evaluating all the $O(n^p)$ possible combinations of p values facility locations. Obviously, this evaluation is not realistic in terms of CPU time, even if the p and n were moderate, and hence more sophisticated methods based on the set covering problem are required, (see Daskin, 1995 and Salhi and Al-khedhairi, 2010). For variable values of p, Kariv and Hakimi (1979) proved that in a general graph both the continuous and the discrete p-centre problems are NP-hard. See also Megiddo and Supowit (1984) for the proof of the NP-hardness of these problems and other related ones.

Solving the ILP model defined by (1.1) – (1.6) using Cplex proved to be inefficient when $n \geq 100$ and $p \geq 10$ taking more than 5 hours of CPU time without guaranteeing optimality, see Alharbi (2010). The author also introduced some reduction schemes to reduce

the CPU time but though some improvements were obtained, the model was still requiring an excessive amount of computational time. These tests were carried out on a PC with 1.5 GHz processor and 512 MB of RAM. Other methods, as will be described next, are used more efficiently to tackle this problem.

Set Covering-based approaches

The objective function of this approach is to find the locations of the minimum set of p facilities such that each demand point is covered within a given specified distance or time (service standard). Minieka (1970) suggested a basic algorithm that depends on solving a finite sequence of Set Covering Problems (SCP). The idea is to choose a covering distance as a radius and to check whether all demand points are covered within this radius using no more than p facilities. The mathematical formulation and the solution method adopted will be provided in chapter 5.

Daskin (1995) developed an algorithm based on Minieka's idea for solving the unweighted vertex p-centre problem using the bisection method that reduces the gap between the upper and lower bounds of the optimal solution. This technique will be adapted in chapter 5 to find the optimal discrete solution which will be used as an initial solution for the continuous problem.

Daskin (2000) studied the problem again when he proposed a new approach based on Lagrangian relaxation. This method was interesting but was not as competitive as the others that are given next.

Ilhan and Pinar (2001) introduced an exact method for solving the vertex p-centre problem optimally, which consists of two phases, namely the LP-Phase and the IP-Phase. In the first phase, they computed a lower bound to the optimal solution of the problem by solving a series of feasibility problems based on an LP formulation for a successive value of R until the LP problem is feasible. In the second phase, starting from the last value of R they also used feasibility problems to check if it is possible or not to serve all demand points without increasing the number of facilities (p). The idea is that if the solution is not feasible in the first phase, which is very fast to perform, then there is no need to go for the second phase using such a value of R. When, the solution is feasible in phase 1, then the value of R (say R_0) is used as a lower bound for the covering radius in the second phase. If

the solution is infeasible for the specific radius R then the value of R is increased until feasibility is reached, where the current value of R refers to the optimal solution of the vertex p-centre problem.

Ilhan *et al*. (2002) developed a simple and efficient exact algorithm for the vertex p-centre problem, which utilises a set covering sub problem for the solution of the vertex p-centre problem. The algorithm is also similar to the algorithm described by Daskin (1995), however this algorithm utilises different sub problems, rather than using a specific sub problem. Here, at each iteration, the authors set a threshold coverage distance as a radius and check whether it is possible to cover all demand pointes with p or with less facilities within this radius, and then update the lower (L) and the upper (U) bounds on the optimal radius. In brief, the initial L and U values are selected first and an appropriate set covering problem is solved by using $R = \frac{L+U}{2}$. Then a check to see whether p or less facilities can cover all demand points within this radius is performed. If yes, they reset U to the coverage distance, otherwise they reset L to the coverage distance. If $L = U$, the search stops; otherwise they set the coverage distance $R = \frac{L+U}{2}$ and then this process is again repeated.

Elloumi *et al*. (2004) used Minieka's idea to solve the problem through the use of a greedy heuristic and the IP formulation of the sub-problem.

Al-Khedhairi and Salhi (2005) proposed some modifications to the Daskin algorithm (1995) and to the one provided by Ilhan and Pinar (2001). In the first approach, they proposed tighter initial lower and upper bounds, and a more appropriate binary search method to decrease the number of sub-problems to be solved. They used the p^{th} minimum value in the distance matrix $(d_{ij})_{ij}$ as the lower bound instead of zero, and they set the upper bound as $U = \underset{i \in I}{Min}\ \underset{j \in J}{Max}\, d_{ij}$. Moreover, they used the Golden Section method as a binary search instead of the bisection method to tighten the upper and lower bounds. In Ilhan and Pinar (2001), the authors also introduced modifications of some steps by using jumps in the updating of R to reduce the number of ILP iterations needed to find the optimal solution while Phase I was kept unchanged.

Recently Salhi and Al-khedhairi (2010) improved this implementation further by using a tighter initial upper bound (U) as the solution of an efficient heuristic. They also derived lower bound (L) accordingly where $L = \alpha\ U$ ($\alpha \cong 0.80$). Note that if L dose not yield a lower bound, L becomes U and the process continues until a proper range of (L, UB) is found. These two bounds were then used within a bisection method as in Daskin's algorithm. It was found that the optimal solutions could be obtained with relatively fewer iterations. A simple but effective implementation of this set covering-based method will be adapted for the new reformulation local search (RLS) when solving the discrete problem which will be described in chapter 5.

1.6 Summary

In this chapter, a brief introduction to location theory and the definition of the research problem were first given. This was followed by applications for the *p*-centre problem and the meta-heuristics that will be used in our research. This chapter also covered the review on the continuous *p*-centre problem while emphasizing both the single and the multi-facility cases. In addition, a brief recap on the techniques used for the vertex *p*-centre problem was provided as some of these methods will be incorporated into our approaches in this research.

In the next chapter, we will concentrate on exploring the possibility of speeding up the implementation of Elzinga-Hearn algorithm, which is the basis of most local searches that will be used when solving the planar *p*-centre problem in subsequent chapters.

Chapter 2

Enhancements for the solution of the continuous *1*-centre problem

2.1 Introduction

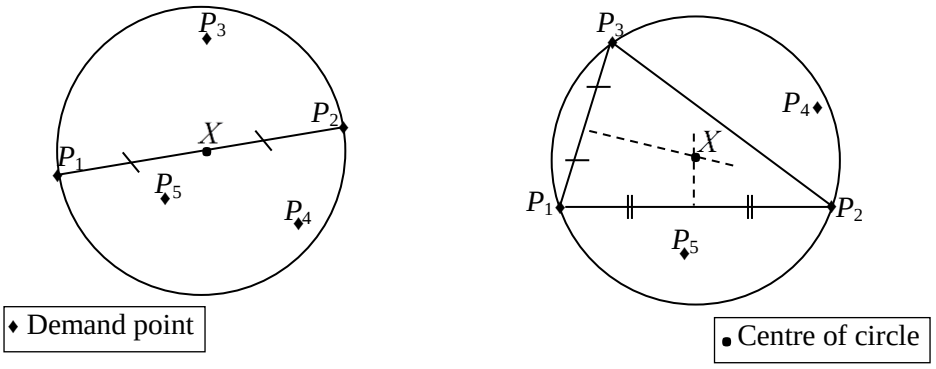
In this chapter, a review of the well known Elzinga-Hearn optimal algorithm for the *1*-centre problem on the plane is first given. Though the algorithm is polynomial, speed up procedures are still worthwhile as our aim is to solve the *p*-centre problem by resolving the *1*-center problem a large number of times. Our proposed enhancements produced a considerable saving when tested on several random instances with various sizes. In addition, we present computational results of the Multi-Start method for the continuous *p*-centre problem to demonstrate the effects of our enhancements against the original Elzinga-Hearn implementation.

2.2 The Elzinga-Hearn algorithm for the unweighted case

As we mentioned in chapter 1, in 1972, Elzinga and Hearn discovered geometrical solutions for some minimax location problems. They considered four closely related minimax location problems. Each involves locating a point in the plane to minimise the maximum distance plus a possible constant k_i (i=1,…,n) to a finite set of points (P_i, i=1,…,n). The four distinct variations were generated by either restricting all of the k_i to be zero, or letting all (or some) of them be positive. The first case is one facility minimax location with the unweighted Euclidean distance, which is called the Euclidean Delivery Boy Problem. Here, the constants k_i are all equal to zero ($k_i = 0$). For example, this represents locating a site for an emergency helicopter that serves all the demand points so that it is as close as possible to the farthest point. The second case is the Euclidean Messenger Boy Problem and here $k_i > 0$ for some point P_i (or all). This is an extension of the first one where

the helicopter travels first to the point P_i and then to some other place (e.g., a hospital) situated at a distance k_i away. Here, the messenger boy is located at the point X in order to move from X to a point P_i and then deliver a message a distance k_i away with the aim of minimising the maximum total Euclidean distance. The third case is one facility minimax location with the unweighted Rectilinear distance (the Rectilinear Delivery Boy Problem) whereas the last one is about the Rectilinear Messenger Boy Problem that has the same interpretation except that the emergency vehicle is restricted to traveling on a grid.

Here, the objective of the Elzinga-Hearn algorithm (*1*-centre location problem) is to minimise the maximum distance between a customer and its nearest facility. This problem has a succinct geometrical interpretation, which is to find the smallest circle that encloses a given set of *n* points (customers). The centre of this circle is precisely the location of the new facility, as shown in Figure 2.1. Note that any circle can be determined by two or three points.



a) Determined by two points *b*) Determined by three points

Figure 2.1: The circle can be determined by two points or by three points

It can be noted that in Figure 2.1 (*a*), the minimum covering circle is determined by two points (P_1 and P_2) that are at the two ends of the diameter where its centre is at the midpoint of points P_1 and P_2. In Figure 2.1 (*b*), the minimum covering circle is determined by three points (P_1, P_2, and P_3) that are on its circumference (edges of an acute triangle) with its centre being at the circumcenter of the triangle (P_1 P_2 P_3).

The idea of this algorithm is to construct successively larger and larger circles, defined by two or three points, until the optimum is attained. The approach is formally given next.

The Elzinga-Hearn algorithm

Step 1: Select two random points, P_s and P_t

Step 2: Construct the circle whose diameter is $d(P_s, P_t)$.

a) If this circle includes all points, then the centre of the circle is the optimal solution X and stop.

b) Else, select a point P_u outside the circle.

Step 3: If the triangle determined by P_s, P_t and P_u is an obtuse or a right angled triangle, rename the two points on the two ends of the hypotenuse as P_s and P_t and go to *Step* 2.

Step 4: Else, construct the circle passing through these three points. (The centre of the circle is at the intersection of the perpendicular bisectors of two sides of the triangle.) If the circle includes all the points, then the centre of the circle is the optimal solution X and stop, else go to *Step* 5.

Step 5: Select a point P_v not enclosed by the circle and let $Q = Arg\ Max\ \left(d(P_v, P_s), d(P_v, P_t), d(P_v, P_u)\right)$ be the point among $\{P_s, P_t, P_u\}$ that is farthest from P_v. Extend the diameter through the point Q to a line that divides the plane into two half planes. Let the point L be the point among $\{P_s, P_t, P_u\}$ that is in the half plane opposite P_v. Go to *Step* 3 using the three points Q, L, and P_v. See Figure 2.2 for an illustration.

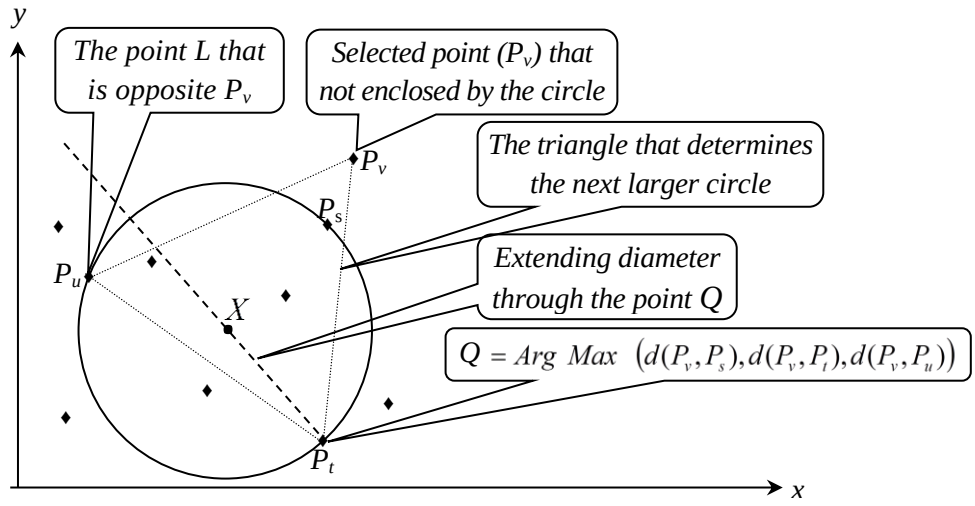


Figure 2.2: Choose the three points among the four points (P_s, P_t, P_u and P_v) in *Step* 5

***Note*:** In *Step* 5, the point P_u could not be on the diameter. This is because if this point (P_u) was on the diameter, it means that the circle is determined by two points (P_t and P_u), where its centre (X) would have been already found in *Step* 3.

2.3 Enhancements to the Elzinga-Hearn algorithm for the unweighted case

Elzinga and Hearn (1972) noted that their algorithm has two weaknesses: (i) selection of the starting points (*Step* 1) and (ii) the selection of the uncovered points (*Step* 2 (*b*) and *Step 5*), which will be explained later. This weakness gave rise to the design of six enhancements which are given next.

2.3.1 Variant (1): *V1*

The steps of this variant (*V1*) are similar to the steps of the original algorithm, except that *Step* 1, is modified and *Step* 3 of the original algorithm is not needed here. Thus, the steps of *V1* are as follows including those unchanged steps given here for convenience.

Step 1: Select the two starting points, P_s and P_t that have the greatest distance between any two points. Mathematically $(P_s, P_t) = Arg\ \ Max_{\substack{i=1,\dots,n \\ j=1,\dots,n}}\ d(P_i, P_j)$, (see Figure 2.3).

Step 2: Construct the circle whose diameter is $d(P_s, P_t)$.

a) If this circle covers all points, then the centre of the circle is the optimal solution X and stop.

b) Else, select a point P_u which is not enclosed by the circle.

Step 3: Construct the circle passing through these three points (P_s, P_t and P_u). If the circle covers all the points, then the centre of the circle is the optimal solution X and stop, else go to *Step* 4.

Step 4: Select a point P_v not enclosed by the circle, and let Q be the point among $\{P_s, P_t, P_u\}$ that is farthest from P_v. Extend the diameter through the point Q to a line that divides the plane into two half planes. Let the point L be the point among $\{P_s, P_t, P_u\}$ that is in the half plane opposite P_v. Go to *Step* 3 using the three points Q, L, and P_v.

***Note** 1:* In this variant (*V1*), P_s and P_t in *Step* 1 are selected as those farthest apart. *Step* 3 of the original algorithm is not needed, because the use of the two points farthest apart as initial starting points (*Step* 1) guarantees that all the subsequent circles will be defined by three points. In other words, any uncovered point with the farthest two points (P_1, P_2) makes an acute triangle (see Figure 2.3). This means that we do not need to check the case when the circle is defined by 2 critical points, (see Figure 2.3). For instance, if the points P_1 and P_2 are the farthest two points, this means that there are no points in the region (*a*) and (*c*). Furthermore, any point in region (*b*) which is not encompassed by the circle, will make an acute triangle with the points P_1 and P_2. This is based on Thales' theorem, as shown in Appendix *A1*.

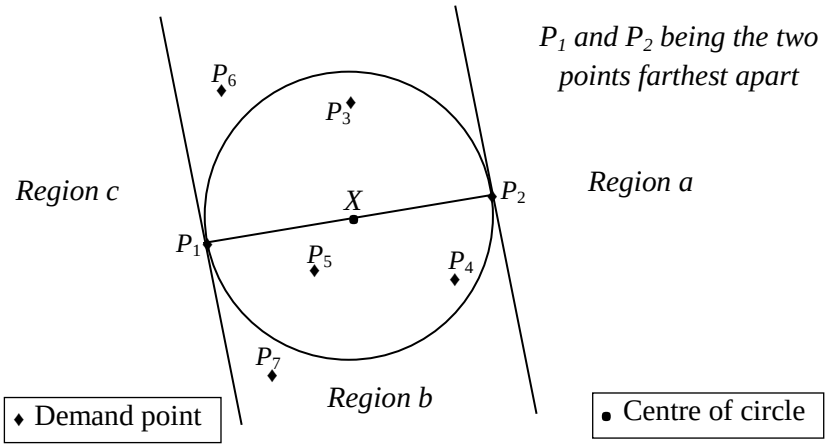


Figure 2.3: Illustration of the farthest points (P_1 and P_2) and the corresponding regions

Based on the above observation the use of the two points farthest apart (P_1, P_2) as initial starting points could lead to reducing the number of iterations significantly. On the other hand, to find these two points, the Euclidean distance should be calculated $(n^2 - n)/2$ times. The question is therefore:

Is the time that we can save from reducing the number of iterations and checking the type of triangles (acute or not) less than the time required to find the farthest two points?

The answer to this will be provided empirically in our computational results section.

Note 2: We can reduce the time required to find the farthest two points by finding the maximum without using the square root as $Arg\ Max_{i,j}\left((x_i - x_j)^2 + (y_i - y_j)^2\right) =$

$Arg\ Max_{i,j}\ \sqrt[2]{(x_i - x_j)^2 + (y_i - y_j)^2}$. Though the saving by not computing the square root is very small for one pair of points, here this needs to be performed $\frac{n^2 - n}{2}$ times which can become not negligible.

2.3.2 Variant (2): *V2*

The steps of the second variant (*V2*) are the same as the steps of *V1* except that *Step* 2 (*b*) and *Step* 4 of *V1*are modified. The steps of *V2* are summarised as follows:

Step 1: Select the two farthest points, P_s and P_t as in *V1*.

Step 2: Construct the circle whose diameter is $d(P_s, P_t)$.

a) If this circle covers all points, then the centre of the circle is the optimal solution *X* and stop.

b) Else, select a point P_u that has the greatest distance from the previous solution (midpoint of P_s and P_t).

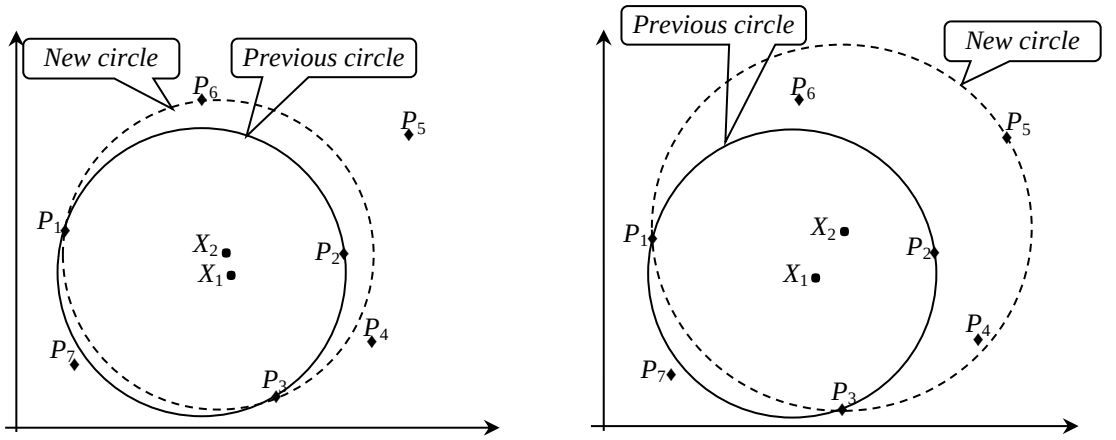
Step 3: Construct the circle passing through these three points (P_s, P_t and P_u). If the circle covers all the points, record the centre of the circle (the optimal solution *X*) and stop, else go to *Step* 4.

Step 4: Select a point P_v that has the greatest distance from the previous solution *X* and let *Q* be the point among $\{P_s, P_t, P_u\}$ that is farthest from P_v. Extend the diameter through the point *Q* to a line that divides the plane into two half planes. Let the point *L* be the point among $\{P_s, P_t, P_u\}$ that is in the half plane opposite P_v. Go to *Step* 3 using the three points *Q*, *L*, and P_v.

The idea of *Step* 2 (*b*) and *Step* 4 are also proposed by Elzinga and Hearn, where they mentioned: *"In practice it seems reasonable to choose the farthest point outside the current circle rather than just some point as indicated in Steps 2 and 3."* (Elzinga and Hearn, 1972, p. 382). However, they neither applied this idea nor tested it. Furthermore, Hearn and Vijay (1982) mentioned: *"The speed of this method depends on whether the first two points are chosen randomly or by some heuristic, and on whether the outside point of Step 3 is chosen to be the farthest point or randomly."* (Hearn and Vijay, 1982, p. 794). Their experiments are based on a random point outside and a farthest point outside the current circle with a random

start and a heuristic start. In their limited experiment, they found that random starts are slightly faster. In the same work for the weighted case, Hearn and Vijay also used the idea of the farthest point from the current solution (weighted-farthest point). However, our results are different. This aspect will be further explained in our computational results section.

Choosing the farthest point that is not covered by the current circle leads to reducing the number of iterations. This is because the choice of the farthest point from the centre of the circle will create a new larger circle that covers most of the points, see Figure 2.4.



(*a*) Choose randomly a point outside the circle (*b*) Choose the farthest point outside the circle

Figure 2.4: The choice of any point or a farthest point outside the current circle

If we choose any random point not encompassed by the circle (let it be P_6) as in Figure 2.4 (*a*), the new circle is determined by three points (P_1, P_3 and P_6). When we compare this circle with the previous circle (its centre X_1), we note that it did not cover any additional point except (P_6). However, when we choose the farthest point outside the current circle as in Figure 2.4 (*b*) the new circle is determined by three points (P_1, P_3 and P_5), which covers all the points except point P_7. On the other hand, to find the farthest point outside the current circle, the Euclidean distance should be calculated *n* times. The efficiency of this variant, known as *V2*, will be shown in our computational results section.

2.3.3 Variant (3): *V3*

Here, we will use the idea initially proposed by Bass and Schubert (1967) to improve Sylvester's algorithm (1857), where the two initial starting points are chosen on the basis of

extreme values of the coordinates. The steps of this variant (*V3*) are similar to the steps of the original algorithm, except that *Step* 1, which is modified as follows:

Step 1: Choose four points that determine the maximum and the minimum point in the x-axis and the y-axis. Let $B = \{i_1, i_2, j_1, j_2\}$ with $i_1 = Arg\ Min_i(x_i)$, $i_2 = Arg\ Max_i(x_i)$, $j_1 = Arg\ Min_j(y_j)$ and $j_2 = Arg\ Max_j(y_j)$. Choose $P_s \in B$ and $P_t \in B$ such that $(P_s, P_t) = Arg \underset{\substack{i \neq j \\ (P_i, P_j) \in B \times B}}{Max} d(P_i, P_j)$.

Other steps are unchanged.

These four chosen points namely i_1, i_2, j_1 and j_2 are illustrated in Figure 2.5.

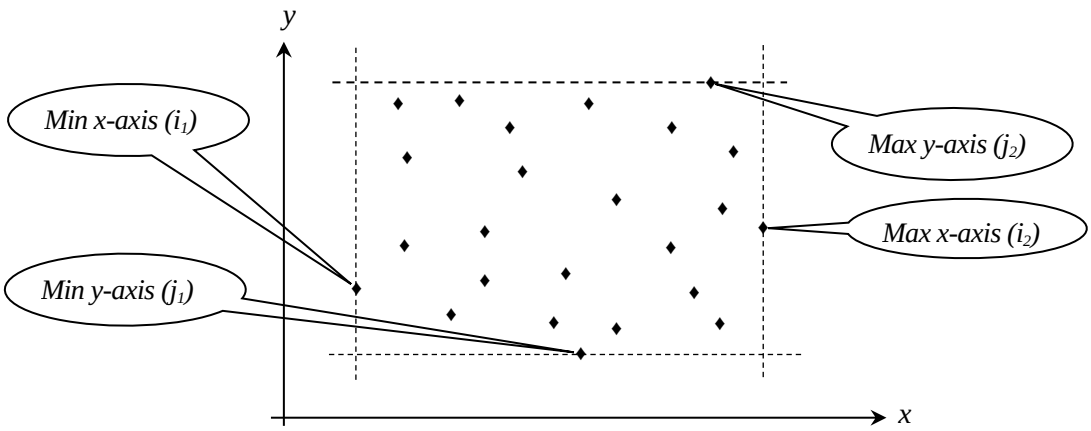


Figure 2.5: The four extreme points with the max and the min x-axis and the y-axis

Note that, using the two points that have the greatest distance between those four points (max and min of x-axis and y-axis) as the initial starting points does not necessarily cover all the points, as shown in Figure 2.6.

Using the farthest two points of the four points (max and min of x-axis and y-axis) as the initial starting points leads to reducing the number of iterations significantly. On the other hand, finding these two points requires an extra number of calculations, as the largest and the smallest value of x-axis and y-axis need to be found, and then the two points that have the greatest distance among these four points will also need to be determined. Our computational results will demonstrate the effectiveness of this variant which we refer to as *V3*.

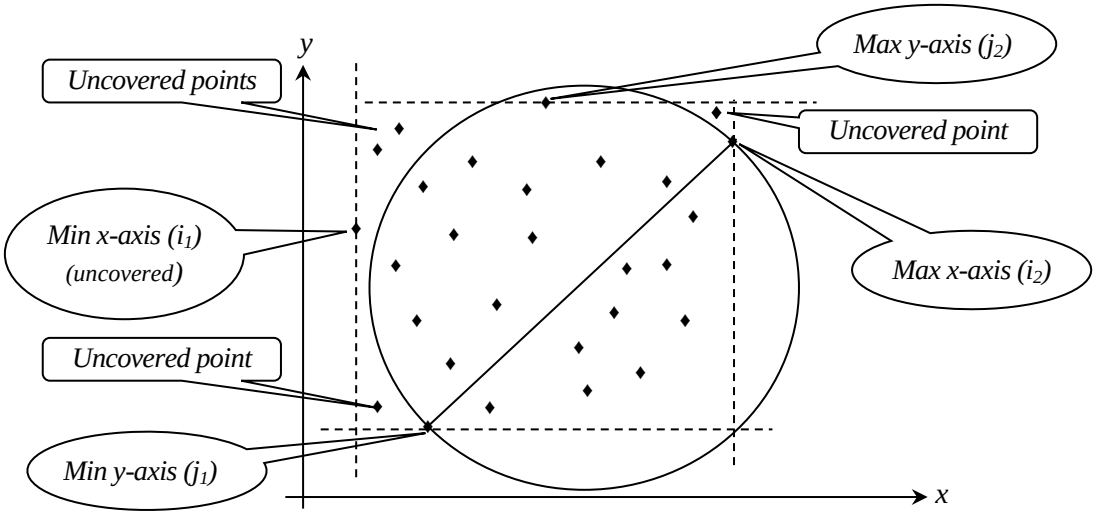


Figure 2.6: The two farthest points among the four points (max and min of x-axis and y-axis) not covering all points

2.3.4 Variant (4): *V4*

All the steps of this variant (*V4*) are the same as the steps of the original algorithm except that *Steps* 1, 2 (*b*) and 5 are modified. The steps of this variant are as follows:

Step 1: As *Step* 1 in *V3*.

Step 2: Construct the circle whose diameter is $d(P_s, P_t)$ and centre *X*.

a) If the circle covers all points, then its centre is the optimal solution *X* and stop.

b) Else, select a point $P_u = Arg \underset{i=1,\ldots,n}{Max}\, d(P_i, X)$.

Step 3: If the triangle determined by P_s, P_t and P_u is an obtuse or a right triangle, rename the two points on the two ends of the hypotenuse as P_s and P_t and go to *Step* 2.

Step 4: Else, construct the circle passing through these three points and let *X* be its centre. If the circle covers all the points, stop, else go to *Step* 5.

Step 5: Select a point P_v not enclosed in the circle such that $P_v = Arg \underset{i=1,\ldots,n}{Max}\, d(P_i, X)$. Let *Q* be the point among $\{P_s, P_t, P_u\}$ that is farthest from P_v. Extend the diameter through the point *Q* to a line that divides the plane into two half planes. Let the point *L* be the point among $\{P_s, P_t, P_u\}$ that is in the half plane opposite P_v. Go to *Step* 3 using the three points *Q*, *L*, and P_v.

$$Q = Arg\ Max\ \left(d(P_v, P_s), d(P_v, P_t), d(P_v, P_u)\right)$$

We will test the efficiency of this variant in the computational results section. This enhancement, known as *V4* can be considered as a combination of *V2* and *V3*.

2.3.5 Variant (5): *V5*

The steps of *V5* are similar to the steps of the original algorithm, except that *Steps* 1 and 2, are modified. In addition, *Step* 3 of the original algorithm is not needed here. The steps of this variant are as follows:

Step 1: Select the two farthest points, P_s and P_t as in *V1*.

Step 2: Construct the circle whose diameter is $d(P_s, P_t)$.

a) If the circle encloses all points, then its centre is the optimal solution *X* and stop.

b) Else select a point P_u that has the greatest distance between the two points P_s and P_t, (as shown in Figure 2.7).

Step 3: Construct the circle passing through these three points (P_s, P_t and P_u). If the circle covers all the points, then its centre is the optimal solution *X* and stop, else go to *Step* 4.

Step 4: Select a point P_v not enclosed by the circle, and let *Q* be the point among $\{P_s, P_t, P_u\}$ that is farthest from P_v. Extend the diameter through the point *Q* to a line that divides the plane into two half planes. Let the point *L* be the point among $\{P_s, P_t, P_u\}$ that is in the half plane opposite P_v. Go to *Step* 3 using the three points *Q*, *L*, and P_v.

$$Q = Arg\ \ Max\ \ \left(d(P_v, P_s), d(P_v, P_t), d(P_v, P_u)\right)$$

Here we extend *V1* by choosing the third point which is farthest from the other two points having the greatest sum distance, as shown in Figure 2.7. In other words, $P_u = Arg\ Max_{\substack{P \neq Ps \\ P \neq Pt}} \left(d(P, P_s) + d(P, P_t)\right)$. This leads to reducing the number of iterations.

However, finding these three points requires an extra number of calculations. This needs finding the farthest two points (the Euclidean distance (without using square root) should be calculated $(n^2 - n)/2$ times) as well as the third point (P_u) which requires checking $\left(d(P, P_s) + d(P, P_t)\right)$ in the distance matrix $(n-2)$ times. The gain in computation time against the extra computational burden for this variant, known as *V5*, will be shown in our computational results section.

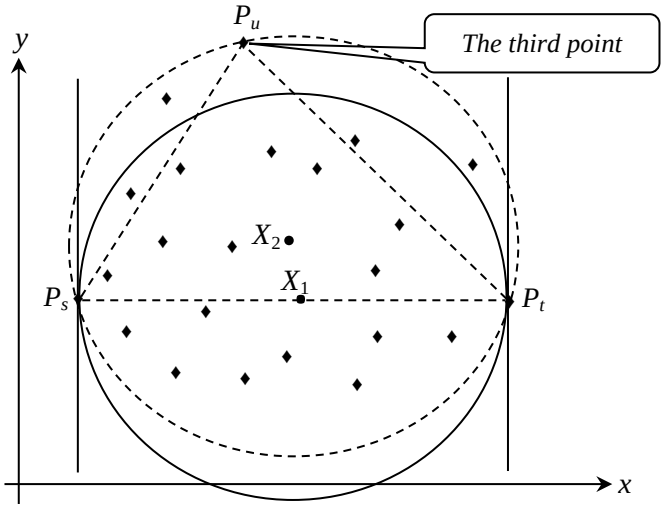


Figure 2.7: The third point with the farthest two points (P_s and P_t)

2.3.6 Variant (6): *V6*

The steps of this variant (*V6*) are the same as those of the previous variant (*V5*) except that *Step* 4 is modified as follows:

Step 4: Select a point P_v not enclosed in the circle such that $P_v = Arg \max_{i=1,\ldots,n} d(P_i, X)$. Let Q be the point among $\{P_s, P_t, P_u\}$ that is farthest from P_v. Extend the diameter through the point Q to a line that divides the plane into two half planes. Let the point L be the point among $\{P_s, P_t, P_u\}$ that is in the half plane opposite P_v. Go to *Step* 3 using the three points Q, L, and P_v.

This variant is a combination of *V2* and *V5* and its efficiency will be presented in our computational results section.

2.4 The Elzinga-Hearn algorithm for the weighted case

Here, the objective of the Elzinga-Hearn algorithm is to minimise the maximum weighted Euclidean distance between a demand point and its nearest facility.

Let $w_i > 0$ be a positive constant weight applied to a demand point i, $i = 1, \ldots, n$.

All the other notation is unchanged.

The objective becomes:

Minimise $f(X) = \max\{ w_i\, d(X, P_i): i = 1, \ldots, n \}$.

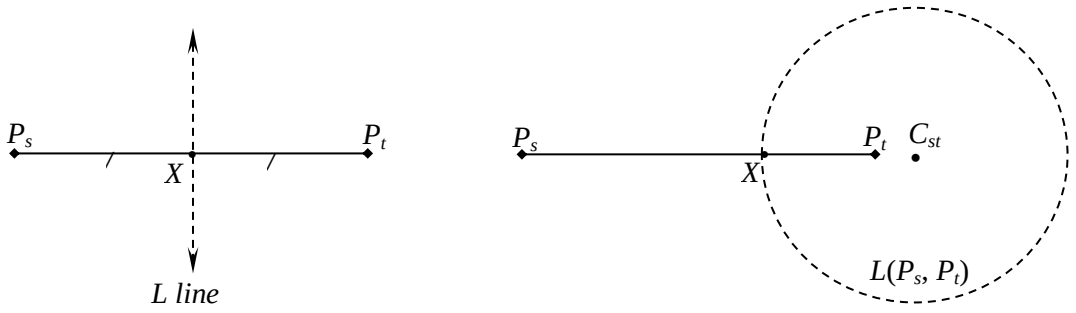
$X \in \Re^2$.

The optimal solution is guaranteed using a geometrical approach based on the following interesting results. It can be shown that the optimal solution for the weighted minimax location problem can be determined by one, two or three points only. These are usually referred to in the literature as the critical points.

***Result 1* (*Case of 2 points P_s and P_t*):**

For 2 points P_s and P_t, let $L(P_s, P_t) = \{ X : w_s\, d(X, P_s) = w_t\, d(X, P_t)\}$, that is, $L(P_s, P_t)$ is the set of points whose weighted distance to point P_s equals the weighted distance to point P_t . If the ratio $r = \frac{w_s}{w_t} = 1$, then $L(P_s, P_t)$ reduces to a straight line, i.e., the perpendicular bisector of the line joining point P_s and point P_t. If $r \neq 1$, then $L(P_s, P_t)$ is a circle with radius $\frac{rl_2(p_s, p_t)}{|1-r^2|}$ and centre $\frac{p_s - r^2 p_t}{1-r^2}$ known as the Apollonius circle. Figure 2.8 shows the set L for the unweighted and the weighted cases of two points (P_s, P_t).

Figure 2.8 (*a*) shows the feasible solutions when all weights are equal (unweighted case), which are the set of points $L(P_s, P_t)$ that are located on the line L. In other words, for any point of $L(P_s, P_t)$ the distance from this point to the point P_s is equal the distance from that point to the point P_t. However, the optimal solution is at point X, because at this point the distance (from X to P_s and from X to P_t) is the minimum distance. In Figure 2.8 (*b*), the same idea can be applied when $r \neq 1$ (weighted case), however the set of points $L(P_s, P_t)$ is now a circle, and the weighted distance is used. Thus the optimal solution is at the point X, where the circle and the line $P_s\ P_t$ intersect.



(*a*) Solutions of the unweighted case (*b*) Solutions of the weighted case

Figure 2.8: The set L for the unweighted and the weighted case of 2 points

Result 2 (case of 3 points):

This result is used to find the optimal location of the weighted case (*1*-centre problem) of 3 points (n=3), where the optimal solution can be determined by two or three points. In other words, the optimal solution of the location problem with 3 points P_s, P_t, and P_u is determined by one of the pair of points: P_s and P_t $\{a=X(P_s, P_t)\}$, or P_s and P_u $\{b=X(P_s, P_u)\}$, or P_t and P_u $\{c=X(P_t, P_u)\}$, or the optimal solution is determined by all three points in which case the solution lies at the intersection of $L(P_s, P_t)$, $L(P_s, P_u)$ and $L(P_t, P_u)$ $\{d_1, d_2\}$. Namely, if there is an intersection between all the three circles ($L(P_s, P_t)$, $L(P_s, P_u)$ and $L(P_t, P_u)$), the intersection points will be at two points only (d_1 and d_2)**.** Figure 2.9 displays these five candidate points.

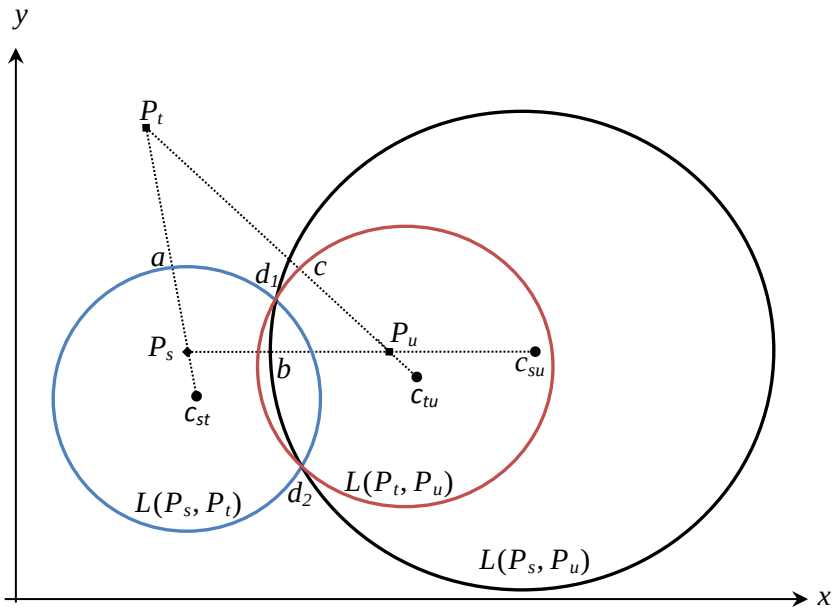


Figure 2.9: The optimal solution for the 3 points (P_t, P_s, P_u)

From the five possibilities $\{a, b, c, d_1, d_2\}$, the choice will reduce to choosing one point from $\{d_1, d_2\}$ or from $\{a, b, c\}$. In other words, the optimal location can be either at the two intersection points between all the three circles $\{d_1, d_2\}$ as shown in Figure 2.9 or there would be no intersection at all as illustrated in Figure 2.10. This means that, the solution cannot be at the intersection of two circles only. The proof of this is given in Appendix *A2*.

(i) Case when there is no intersection

If there is no intersection between $L(P_s, P_t)$, $L(P_s, P_u)$ and $L(P_t, P_u)$, the optimal solution can be determined by 2 points only. In other words, the optimal solution can be at $\{a, b \text{ or } c\}$, as shown in Figure 2.10.

Note that the task is straightforward and hence it will not require a heavy extra computational burden.

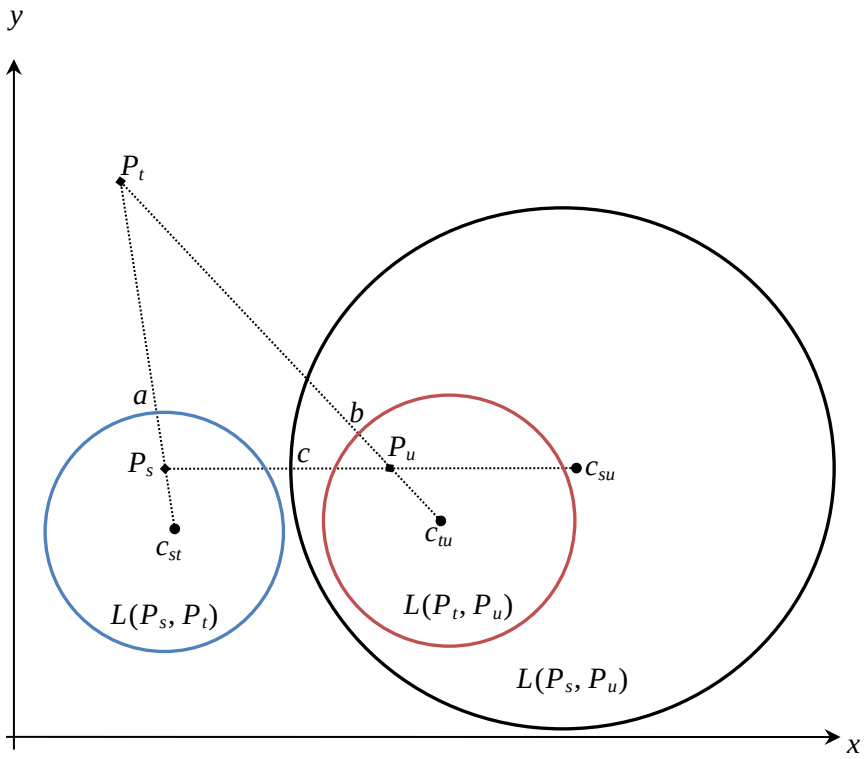


Figure 2.10: Case of no intersection between circles

(ii) Case when there is an intersection

We can classify this case into two types**,** which are as follows:

a) The intersection is outside the triangle

If the intersection between the circles is outside the triangle $P_s P_t P_u$, the optimal solution can be determined by 2 points only as in the previous case (i). The optimal solution can be at one of the three points namely *a*, *b* or *c* as shown in Figure 2.11.

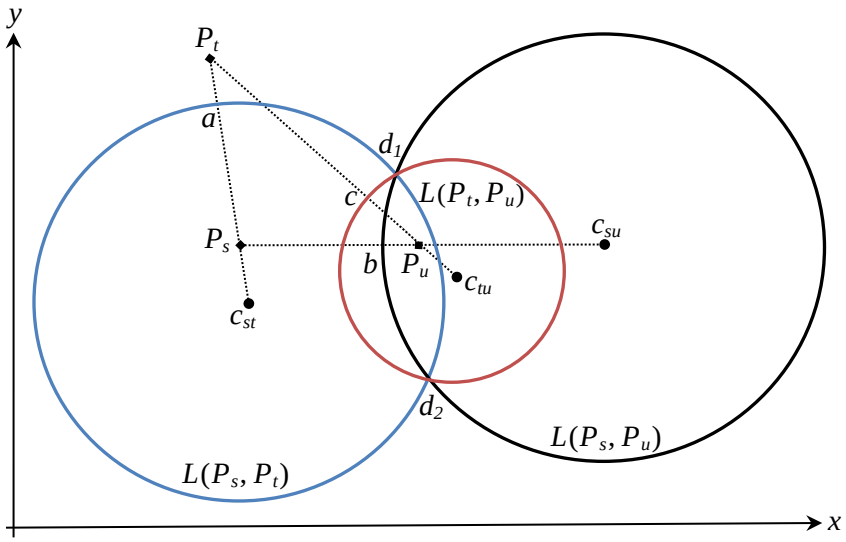


Figure 2.11: The intersection is outside the triangle $P_s\ P_t\ P_u$

b) The intersection is inside the triangle

If the intersection between the circles is inside the triangle $P_s\ P_t\ P_u$ then the optimal solution can be either at points d_1 or d_2 as shown in Figure 2.12.

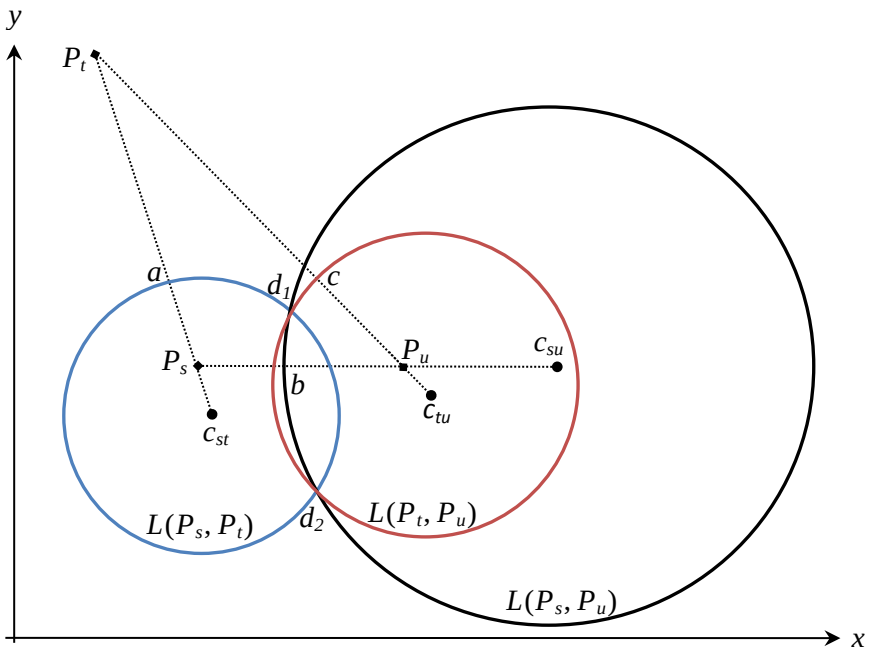


Figure 2.12: The optimal solution for the 3 points (inside the triangle i.e., d_1 or d_2)

Note:

If the intersection of these three circles is outside the triangle as in Figure 2.11, the optimal solution will be defined by two points**.** If this intersection is inside the triangle as in Figure 2.12, the optimal solution will be determined by three points. The proof of this note is as follows:

Proof

In Figure 2.11, the circle with its centre at c_{tu} presents the optimal solution at point c, which is determined by two points P_t and P_u. Here we will prove that this solution (at point c) is better than the solution at point d_1 (determined by the three points P_t, P_u and P_s). In other words, if the intersection of the circles is outside the triangle, the optimal solution will be determined by two points, which means that any solution outside the triangle is not an optimal solution. The optimal solution is either inside the triangle (determined by three points) or on one side of the triangle (determined by two points), which means that the optimal solution cannot be outside the triangle.

In Figure 2.11, the intersection is outside the triangle at d_1 and d_2 and the optimal solution is determined by two points P_t and P_u at point c.

$\because$ both points d_1 and c are located on the circumference of the circle whose centre is at c_{tu}.

$\therefore\ w_t\, d(d_1, P_t) = w_u\, d(d_1, P_u)$ and $w_t\, d(c, P_t) = w_u\, d(c, P_u)$

$\because\ d(d_1, P_t) > d(c, P_t)$ because the points P_u, c and P_t are located on the straight line ($\overline{P_t P_u}$), whereas, the points P_u, d_1 and P_t do not lie on this straight line.

$\therefore$ the solution at point c is therefore better than the one at point d_1. ■

Figure 2.12 shows that the optimal solution at point d_1 is determined by three points P_t, P_u and P_s. Here we will prove that the solution at point d_1 is better than the solution at point c which is determined by two points P_t and P_u.

$w_t\, d(d_1, P_t) = w_u\, d(d_1, P_u) = w_s\, d(d_1, P_s)$, because this point ($d_1$) is the intersection point of all these three circles.

$w_t\, d(c, P_t) = w_u\, d(c, P_u)$, because this point ($c$) is the intersection point of the circle whose centre is at c_{tu} with the line segment $\overline{P_t P_u}$.

$w_t\, d(a,\, P_t) = w_s\, d(a,\, P_s)$, because this point ($a$) is the intersection point of the circle c_{st} with the line segment $\overline{P_t P_s}$.

As the point c is located outside the circle whose centre is at c_{st} and the point d_1 is located on the same circle.

$\therefore\ w_s\, d(c,\, P_s) > w_s\, d(d_1,\, P_s)$, which means that the solution at the point d_1 is better than the one at point c. ■

In order to know whether there is an intersection between the circles or not, we must perform the following checks.

For any two circles on the plane, several situations may occur. (i) There is no intersection between the two, (ii) one of them contains the other or (iii) one of them touches the other from the outside or from the inside. The conditions to guarantee the existence or not of the intersection between two circles is given in Appendix *A3* and the coordinates of the intersection between the two circles, if this exists, is provided in Appendix *A4*.

Based on the above results, the following algorithm was developed by Elzinga and Hearn (1972) to find the optimal location of the weighted *1*-centre problem in the continuous space.

The Elzinga-Hearn algorithm (***Weighted***)

Step 1: Choose any two points randomly P_s and P_t. Solve the weighted minimax location problem with P_s and P_t and let $Z = w_s\, d(X,\, P_s)$, where X is computed by using Result 1.

Step 2: If $w_i\, d(X,\, P_i) \leq Z$ for all P_i (i =1,...,n), stop, else select a point P_u such that $w_u\, d(X,\, P_u) > Z$ and go to *Step* 3.

Step 3: Solve the weighted minimax location problem with P_s, P_t, and P_u, for X and Z using Result 2.

Step 4: If the location X is determined by two points, say P_s and P_t go to *Step* 2.

Else, X is determined by three points. If $w_i\, d(X,\, P_i) \leq Z$ for all P_i, stop, otherwise choose the point P_v such that $w_v\, d(X,\, P_v) > Z$.

Step 5: Using P_s, P_t, P_u and P_v, choose all combinations of two points to find the location X using Result 1, and choose all combinations of three points to find the location X using Result 2.

If X and Z are determined by two points, call them P_s and P_t and go to *Step* 2.

If X and Z are determined by three points, call them P_s, P_t, and P_u, and go to *Step* 5.

2.5 Enhancements to the Elzinga-Hearn algorithm for the weighted case

In this section, we will present six enhancements on the original Elzinga-Hearn algorithm for the weighted case. In general, the modifications are similar to those that have been proposed for the unweighted case.

Variant (1*w*): *W*1

The steps of this variant (*W1*) are the same as the steps of the original algorithm except that *Step* 1 is modified as follows:

Step 1: Choose the two farthest points P_s and P_t that have the greatest weighted distance between any two points. Solve the weighted minimax location problem with P_s and P_t and let X be the solution with $Z = w_s d(X, P_s)$ using Result 1.

The other steps are unchanged.

In *Step* 1, the two points P_s and P_t are defined by $(P_s, P_t) = Arg\ Max_{\substack{i \neq j \\ i,j=1,\ldots,n}} \{\frac{w_i \times w_j}{w_i + w_j} d(P_i, P_j)\}$

Variant (2*w*): *W*2

Here, the steps are similar to those of the original algorithm except that *Steps* 1, 2 and 4 are modified. The steps of this variant are as follows:

Step 1: As *Step* 1 of *W1*.

Step 2: If $w_i d(X, P_i) \leq Z$ for all P_i, stop, else select a point P_u such that $w_u d(X, P_u) > Z$ that has the greatest weighted distance from X and go to *Step* 3.

Step 3: This step is unchanged.

Step 4: If the location X is determined by two points, say P_s and P_t go to *Step* 2,

Else, X is determined by three points. If $w_i\, d(X, P_i) \leq Z$ for all P_i then stop, otherwise choose the point P_v such that $w_v\, d(X, P_v) > Z$ and which has the greatest weighted distance from X.

Step 5: This step is unchanged.

Variant (3*w*): *W3*

All the steps of *W3* are the same as the ones of the original algorithm except that *Step* 1 which is modified as follows:

Step 1: Choose four points that determine the maximum and the minimum point in the x-axis and the y-axis. Let $B = \{i_1, i_2, j_1, j_2\}$ with $i_1 = Arg\ Min_i(w_i x_i)$, $i_2 = Arg\ Max_i(w_i x_i)$, $j_1 = Arg\ Min_j(w_j y_j)$ and $j_2 = Arg\ Max_j(w_j y_j)$. Choose P_s and P_t such that $(P_s, P_t) = Arg \underset{\substack{i \neq j \\ (P_i, P_j) \in B \times B}}{Max} \{\frac{w_i \times w_j}{w_i + w_j} d(P_i, P_j)\}$. Use Result 1 to solve the weighted minimax location problem with P_s and P_t, and let X be the solution with $Z = w_s d(X, P_s)$.

Variant (*4w*): *W4*

This variant (*W4*) is a combination of *W2* and *W3* and its steps are as follows:

Step 1: As *Step* 1 of *W3*.

Step 2: As *Step* 2 of *W2*.

Step 3: As *Step* 3 of the original algorithm.

Step 4: As *Step* 4 of *W2*.

Step 5: As *Step* 5 of the original algorithm.

Variant (*5w*): *W5*

The steps of this variant are the same as those of the original algorithm except that steps 1 and 2 are modified as follows:

Step 1: As *Step* 1 of *W1*.

Step 2: If $w_i\, d(X, P_i) \leq Z$ for all P_i, stop.

Else:

a) In the first iteration select a point P_u that has the greatest weighted distance between the two points P_s and P_t.

b) Else choose a point P_u such that $w_u\, d(X, P_u) > Z$ and go to *Step* 3.

The other steps are unchanged.

Variant (*w*6): *W6*

The steps of *W6* are the same as the ones of *W5* except that *Step* 2 (*b*) and *Step* 4 are modified as follows:

Step 1: As *Step* 1 of *W*5.

Step 2: If $w_i\, d(P_i, X) \leq Z$ for all P_i, stop.

Else:

a) In the first iteration select a point P_u that has the greatest weighted distance between the two points P_s and P_t.

b) Else, choose a point P_u such that $w_u\, d(P_u, X) > Z$ that has the greatest weighted distance from *X* and go to step 3.

Step 3: As *Step* 3 of *W*5.

Step 4: As *Step* 4 of *W2*.

Step 5: As *Step* 5 of *W5*.

2.6 Empirical results for the enhancements

The original algorithm and our enhancements were tested on random instances varying in size from *n*=10 to 100 in increments of 10. For each value of *n*, 100 random instances were tested and average results were reported. The demand points with their *x* and *y* coordinates were generated randomly using a uniform distribution within a square area from 0 to 100. In other words, $x_i \in (0, 100)$ and $y_i \in (0, 100); i = 1,..., n$. For the case of the weighted problem, we also generated the weight (w_i, i=1,…,n), in a uniform range from 1 to 10. The original algorithm and the enhancements were coded in C++ using Visual Studio 2008. A laptop computer with a Intel Core 2 Duo processor, 2.0 GHz CPU and 4G memory was used to conduct these experiments. For simplicity, we referred to *V0* and *W0* as the original algorithms for the unweighted and the weighted case respectively.

2.6.1 Computational results of the unweighted case

According to Table 2.1, we noted that all the enhancements required fewer iterations. The average total number of iterations of *V6* was the lowest number of iterations of 1.089, whereas *V0* required 8.193.

Table 2.1: Average CPU time (in seconds) over 100 instances and number of iterations for n = 10 to 100 (unweighted case)

n	*V0*				*V1*				*V2*				*V3*			
	Average CPU	Iterations of			Average CPU	Iterations of			Average CPU	Iterations of			Average CPU	Iterations of		
		2 pt	3 pt	Total		2 pt	3 pt	Total		2 pt	3 pt	Total		2 pt	3 pt	Total
10	0.00004	0.34	0.80	1.14	0.00003	1	1.01	2.01	0.00002	1	0.80	1.78	0.00002	0.73	1.19	1.92
20	0.00006	0.37	0.84	1.19	0.00004	1	1.08	2.08	0.00005	1	0.84	1.84	0.00003	0.93	0.93	1.86
30	0.00010	3.24	4.76	8	0.00009	1	0.97	1.97	0.00013	1	0.73	1.73	0.00005	1.02	2.47	3.49
40	0.00012	3.66	12	8.80	0.00021	1	0.90	1.90	0.00021	1	0.69	1.68	0.00008	1.02	2.78	3.77
050	0.00015	3.61	5.55	9.16	0.00025	1	0.86	1.86	0.00027	1	0.72	1.72	0.00010	1.11	2.81	3.92
60	0.00029	3.58	5.92	9.50	0.00050	1	0.84	1.84	0.00034	1	0.74	1.74	0.00011	1.27	3.23	4.50
70	0.00019	3.96	6.50	10.46	0.00064	1	0.86	1.86	0.00061	1	0.69	1.69	0.00010	1.37	3.42	4.79
80	0.00037	4.02	7.09	11.11	0.00095	1	0.94	1.94	0.00098	1	0.70	1.70	0.00007	1.43	3.57	5
90	0.00031	4.32	6.85	11.17	0.00093	1	0.91	1.91	0.00097	1	0.69	1.69	0.00007	1.57	3.79	5.36
100	0.00029	4.24	7.16	11.40	0.00121	1	0.87	1.87	0.00128	1	0.65	1.65	0.00008	1.66	3.87	5.53
Average	0.00019	3.13	5.75	8.19	0.00049	1	0.92	1.92	0.00049	1	0.73	1.72	**0.00007**	1.21	2.81	4.01

n	*V4*				*V5*				*V6*			
	Average CPU	Iterations of			Average CPU	Iterations of			Average CPU	Iterations of		
		2 pt	3 pt	Total		2 pt	3 pt	Total		2 pt	3 pt	Total
10	0.00001	0.71	0.97	1.68	0.00002	0.34	0.82	1.16	0.00004	0.34	0.80	1.14
20	0.00003	0.71	0.97	1.68	0.00008	0.36	0.85	1.21	0.00007	0.36	0.84	1.20
30	0.00003	0.86	1.55	2.41	0.00012	0.38	0.73	1.11	0.00016	0.38	0.73	1.11
40	0.00002	0.84	1.70	2.54	0.00017	0.34	0.69	1.03	0.00021	0.34	0.69	1.03
50	0.00006	0.89	1.71	2.60	0.00040	0.36	0.70	1.06	0.00029	0.35	0.72	1.07
60	0.00009	1	1.83	2.83	0.00054	0.35	0.74	1.09	0.00049	0.35	0.74	1.09
70	0.00009	1.04	1.88	2.92	0.00072	0.39	0.70	1.09	0.00061	0.39	0.69	1.08
80	0.00009	1.03	1.98	3.01	0.00082	0.37	0.71	1.08	0.00091	0.37	0.70	1.07
90	0.00012	1.10	1.96	3.06	0.00113	0.37	0.69	1.06	0.00101	0.37	0.69	1.06
100	0.00017	1.22	1.83	3.05	0.00129	0.39	0.65	1.04	0.00128	0.39	0.65	1.04
Average	**0.00007**	0.94	1.64	2.58	0.00053	0.37	0.73	1.093	0.00051	0.36	0.73	**1.089**

2 pt: # iterations when the solution was found by two points (solution determined by a right triangle or an acute triangle)
3 pt: # iterations when the solution was found by three points (solution determined by an obtuse triangle)
Total: total # iterations (2 pt +3 pt)

To illustrate the average CPU time of these enhancements, a line chart was also shown in Figure 2.13.

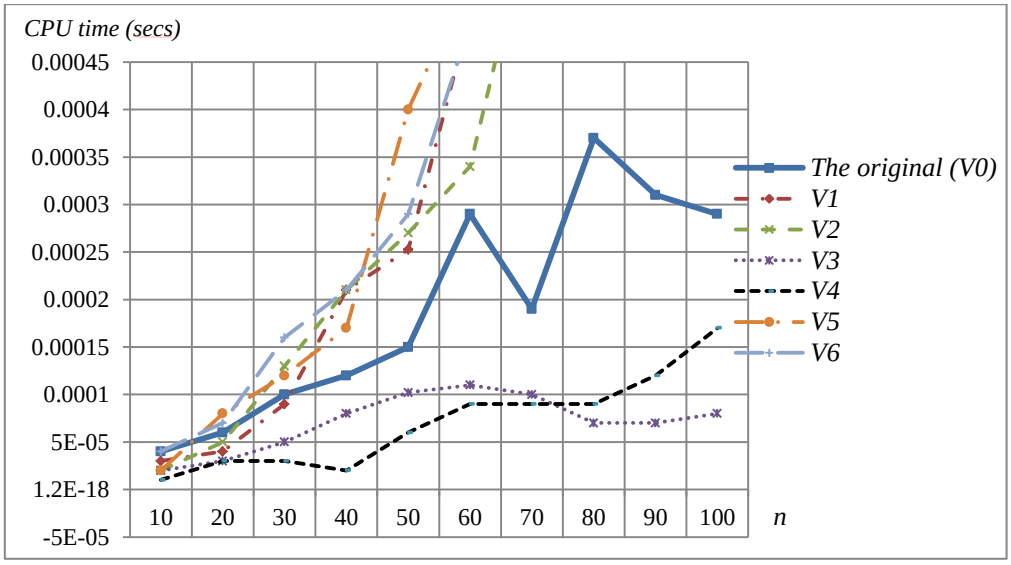


Figure 2.13: Average CPU time of the enhancements over 100 instances for n =10 to 100 (case of unweighted)

We noted that the enhancements *V3* and *V4* were better than the original algorithm and the other enhancements in all cases (from n = 10 to 100). However, *V0* was quicker than *V1*, *V2*, *V5* and *V6* when n was greater than 30. We also noted that there was no clear difference between *V3* and *V4* when n was less than 20. However, when $n < 70$, *V4* was quicker than *V3* and when $n > 70$ the latter became faster than *V4*. In brief, we noted that *V3* and *V4* gave better results than *V0*, especially when n was large.

The following rule can then be used when solving the *1*-centre problem as part of the *p*-centre problem:

If $n \leq 70$ use *V4*, else use *V3*.

To illustrate the differences in CPU time between the original algorithm and the other enhancements, the deviation in % of the enhancements from the original algorithm were computed as follows.

$$Deviation(\%) = \frac{(Time_{Enh} - Time_{Ori})}{Time_{Ori}}.100$$

where $Time_{Enh}$ and $Time_{Ori}$ refer to the CPU time required by the enhancement and the original algorithm respectively.

The results were summarised in Table 2.2 and shown in Figure 2.14.

Table 2.2: Deviation (%) from the original algorithm (*V0*): case of the unweighted

n	The Original Average CPU Time	variant *V1*	variant *V2*	variant *V3*	variant *V4*	variant *V5*	variant *V6*
10	0.00004	-25	-50	-50	-75	-50	0
20	0.00006	-33.33	-16.67	-50	-50	33.33	16.67
30	0.00010	-10	30	-50	-70	20	60
40	0.00012	75	75	-33.33	-83.33	41.67	75
50	0.00015	68.35	80	-31.97	-60	166.67	93.33
60	0.00029	72.41	17.24	-62.07	-68.97	86.21	68.97
70	0.00019	236.84	221.05	-47.37	-52.63	278.95	221.05
80	0.00037	156.76	164.87	-81.08	-75.68	121.62	145.95
90	0.00031	200	212.90	-77.42	-61.29	264.52	225.81
100	0.00029	317.24	341.38	-72.41	-41.38	344.83	341.38
Average	0.00019	105.83	107.58	**-55.57**	**-63.83**	130.78	124.82

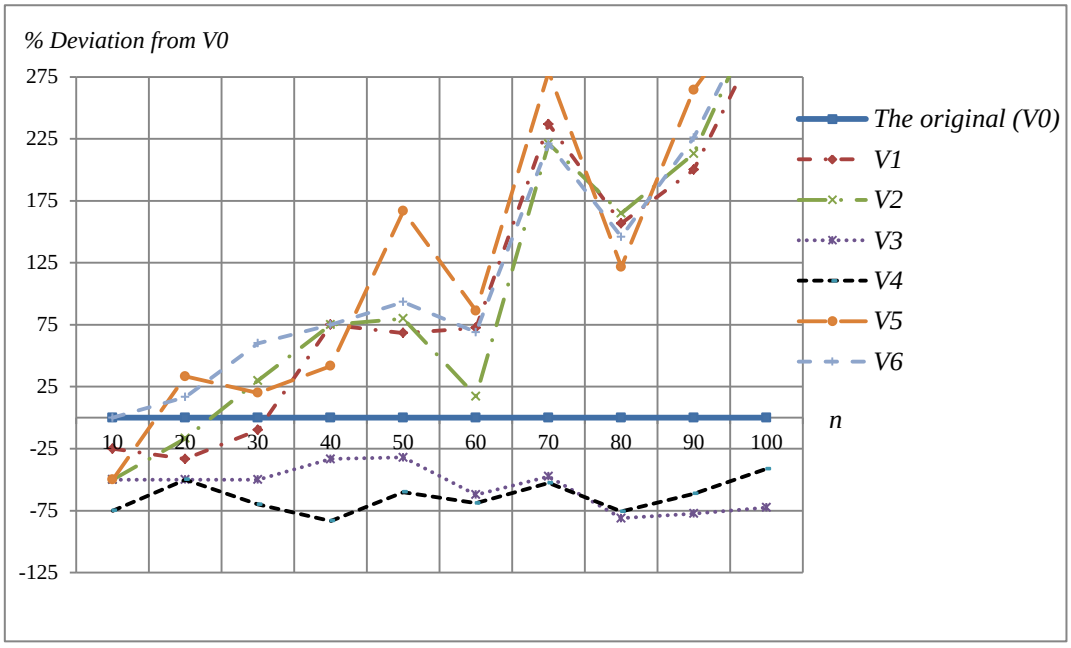


Figure 2.14: Deviation (%) of the CPU time from the original algorithm (unweighted from *n* 10 to 100)

It has been noted that the use of the enhancements (*V3* and *V4*) can save up to 81% and 83% of the time of the original algorithm, when *n* = 80 and 40 respectively. The average saving of about 56% and 64% was recorded for *V3* and *V4* respectively.

Comments:

It is worth noting however that the curve in Figure 2.13 showed some irregularities with respect to the original algorithm when using three points. There was a sharp increase in the CPU time when *n*=60 and *n*=80 compared to the overall trend. On the other hand, there was

also a decrease from n=70 onward. This could be due to the original Elzinga-Hearn algorithm consuming a relatively short time for small values of n. In addition, the amount of computing time also depends on the initial starting points as well as the uncovered point, which in the original algorithm these were randomly selected. To overcome this drawback a large number of extra tests were then performed to better understand this phenomenon.

Additional tests for *V0*, *V3* and *V4*:

Since *V3* and *V4* were the best variants and the differences in their average CPU time when $n \leq 100$ were relatively small, extra data sets (n =50 to 1000) with an increment of 50 were used to better assess the performance of these two variants. These data sets were also used to show the performance of the original algorithm. As before 100 instances were used for each value of n and the averages were recorded. Table 2.3 provides the same information as the one given in Table 2.1.

Table 2.3: Average CPU time and number of iterations for *V0*, *V3* and *V4* over 100 instances for n = 50 to 1000 (case of unweighted)

n	*V0*				*V3*				*V4*			
	Average CPU	Iterations of			Average CPU	Iterations of			Average CPU	Iterations of		
		2 pt	3 pt	Total		2 pt	3 pt	Total		2 pt	3 pt	Total
50	0.00018	3.61	5.61	9.22	0.00007	1.11	2.84	3.95	0.00003	0.89	1.72	2.59
100	0.00026	4.26	6.80	11.06	0.00009	1.66	3.87	5.53	0.00011	1.22	1.83	3.05
150	0.00051	4.83	7.99	12.82	0.00012	2.12	4.82	6.94	0.00016	1.35	1.92	3.27
200	0.00069	5.05	8.83	13.88	0.00012	2.36	5.09	7.45	0.00020	1.44	1.74	3.18
250	0.00087	5.29	9.13	14.42	0.00019	2.59	5.48	8.07	0.00027	1.49	1.78	3.27
300	0.00114	5.35	9.14	14.49	0.00020	2.81	5.82	8.63	0.00036	1.52	1.73	3.25
350	0.00129	5.51	9.74	15.25	0.00028	3.03	6.05	9.08	0.00036	1.63	1.72	3.35
400	0.00154	5.84	9.66	15.50	0.00034	3.17	6.32	9.49	0.00045	1.70	1.72	3.42
450	0.00160	5.90	9.89	15.79	0.00040	3.30	6.61	9.91	0.00051	1.71	1.79	3.50
500	0.00189	5.87	10.68	16.55	0.00046	3.43	6.81	10.24	0.00054	1.73	1.82	3.55
550	0.00222	6.23	9.86	16.09	0.00047	3.46	6.90	10.36	0.00058	1.64	1.94	3.58
600	0.00254	6.49	11.36	17.85	0.00054	3.73	7.35	11.08	0.00062	1.64	2.12	3.76
650	0.00279	6.69	11.31	18	0.00058	3.66	7.36	11.02	0.00069	1.66	2.16	3.82
700	0.00284	6.60	11.38	17.98	0.00063	3.60	7.72	11.32	0.00077	1.54	2.16	3.70
750	0.00313	6.65	11.90	18.55	0.00068	3.18	8.03	11.21	0.00084	1.55	2.05	3.60
800	0.00337	6.53	12.12	18.65	0.00072	3.31	7.68	10.99	0.00096	1.71	1.98	3.69
850	0.00333	6.45	11.70	18.15	0.00078	3.62	7.81	11.43	0.00106	1.73	2.01	3.74
900	0.00360	6.90	11.47	18.37	0.00081	3.89	8.67	12.56	0.00110	1.63	2.12	3.75
950	0.00388	6.79	11.83	18.62	0.00086	3.94	8.80	12.74	0.00123	1.69	2.10	3.79
1000	0.00401	6.58	11.71	18.29	0.00090	4.14	8.53	12.67	0.00123	1.74	2.08	3.82
Average	0.00208	5.87	10.11	15.98	0.00046	3.11	6.63	9.73	0.00060	1.56	1.92	3.48

Summary results from Table 2.3 based on the average CPU time for *V0*, *V3* and *V4* were displayed in Figure 2.15.

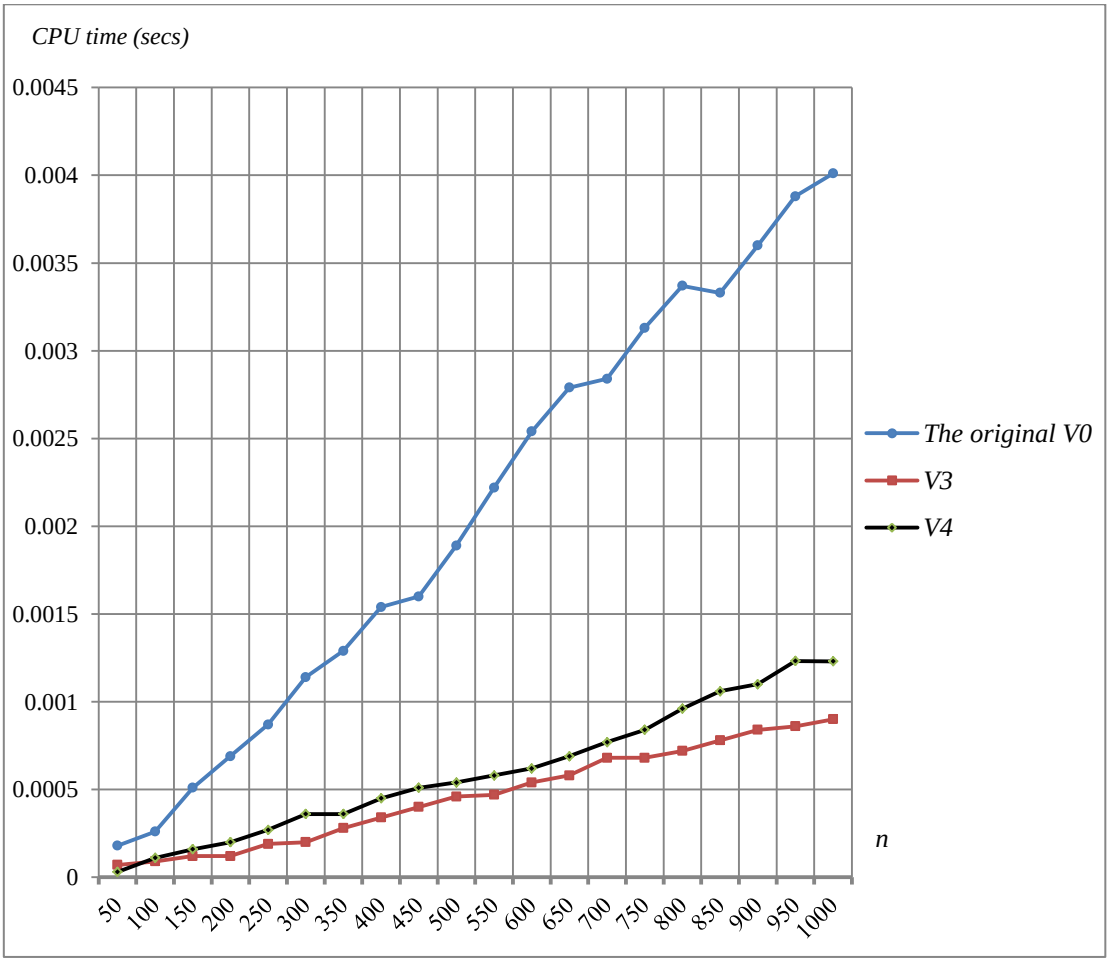


Figure 2.15: Average CPU time of *V0*, *V3* and *V4* over 100 instances from n =50 to 1000 (case of unweighted)

From Figure 2.15, we can conclude that the performances in terms of CPU time of *V3* and *V4* support our claim shown earlier in Table 2.1, confirmed the validity of the previous rule (i.e., if $n \leq 70$ use *V4*, else use *V3*). On the other hand, the differences in the average CPU time between the original algorithm and these two enhancements (*V3* and *V4*) increased even more rapidly with n, where their averages (in seconds) were 0.00208, 0.00046 and 0.00060 respectively. Also the original algorithm showed an overall increase in CPU time with respect to n demonstrating that there was no obvious irregularity as noted in the small data set.

2.6.2 Computational results of the weighted case

According to Table 2.4, it has been observed that all the enhancements required a smaller number of iterations than the original algorithm (*W0*). The averages of the total number of iterations of *W2* and *W6* were the lowest with 2.32, nearly 4 times smaller than *W0* which used 8.21. With respect to the average CPU time, it was found that all the enhancements were faster than the original algorithm when $n \leq 40$. In addition, the enhancements *W3* and *W4* were always quicker than the original algorithm for all values of *n* tested. The average CPU time of these enhancements were also displayed in Figure 2.16.

Table 2.4: Average CPU time and number of iterations over 100 instances for $n = 10$ to 100 (case of weighted)

n	Original Variant (*W0*)				Variant (*W1*)				Variant (*W2*)				Variant (W3)			
	Average CPU	Iterations of			Average CPU	Iterations of			Average CPU	Iterations of			Average CPU	Iterations of		
		2 pt	3 pt	Total		2 pt	3 pt	Total		2 pt	3 pt	Total		2 pt	3 pt	Total
10	0.00020	3.02	2.56	5.58	0.00005	1.42	0.96	2.38	0.00006	1.35	0.89	2.24	0.00007	1.43	0.97	2.40
20	0.00020	3.54	3.28	6.82	0.00014	1.39	1.13	2.52	0.00014	1.32	1.06	2.38	0.00009	1.48	1.22	2.70
30	0.00025	3.89	3.63	7.52	0.00020	1.35	1.09	2.44	0.00020	1.31	1.05	2.36	0.00012	1.63	1.37	3
40	0.00040	4.11	3.89	8	0.00031	1.33	1.10	2.43	0.00028	1.27	1.03	2.30	0.00022	1.70	1.47	3.17
50	0.00052	4.36	4.03	8.39	0.00055	1.45	1.13	2.58	0.00055	1.32	1	2.32	0.00022	1.93	1.61	3.54
60	0.00043	4.59	4.29	8.88	0.00075	1.43	1.14	2.57	0.00049	1.26	0.97	2.23	0.00026	1.96	1.67	3.63
70	0.00058	4.94	4.51	9.40	0.00086	1.45	1.15	2.60	0.00085	1.30	1	2.30	0.00026	2.04	1.74	3.78
80	0.00063	4.58	4.22	8.80	0.00114	1.49	1.15	2.64	0.00097	1.33	1.02	2.35	0.00029	2.09	1.75	3.84
90	0.00090	4.84	4.47	9.31	0.00121	1.46	1.09	2.55	0.00149	1.37	1.03	2.40	0.00029	2.06	1.69	3.75
100	0.00092	4.91	4.55	9.46	0.00162	1.46	1.10	2.56	0.00163	1.32	1	2.32	0.00037	2.08	1.72	3.80
Average	0.00050	4.28	3.94	8.21	0.00068	1.42	1.10	2.53	0.00067	1.32	1.01	**2.32**	0.00022	1.84	1.52	3.36

n	Variant (*W4*)				Variant (*W5*)				Variant (*W6*)			
	Average CPU	Iterations of			Average CPU	Iterations of			Average CPU	Iterations of		
		2 pt	3 pt	Total		2 pt	3 pt	Total		2 pt	3 pt	Total
10	0.00006	1.45	1.01	2.45	0.00012	1.42	0.96	2.38	0.00008	1.35	0.89	2.24
20	0.00007	1.45	1.21	2.66	0.00014	1.39	1.13	2.52	0.00018	1.32	1.06	2.38
30	0.00008	1.54	1.28	2.82	0.00022	1.35	1.09	2.44	0.00018	1.31	1.05	2.36
40	0.00007	1.50	1.26	2.76	0.00039	1.33	1.10	2.43	0.00033	1.26	1.03	2.28
50	0.00015	1.62	1.30	2.92	0.00055	1.45	1.13	2.58	0.00046	1.32	1	2.32
60	0.00013	1.58	1.29	2.87	0.00065	1.44	1.15	2.59	0.00054	1.27	0.98	2.25
70	0.00018	1.62	1.32	2.94	0.00099	1.46	1.16	2.62	0.00086	1.31	1.01	2.32
80	0.00019	1.74	1.40	3.14	0.00122	1.50	1.16	2.66	0.00110	1.34	1.03	2.37
90	0.00027	1.75	1.38	3.13	0.00142	1.47	1.10	2.57	0.00131	1.38	1.04	2.42
100	0.00047	1.71	1.36	3.07	0.00163	1.47	1.11	2.58	0.00146	1.33	1.01	2.34
Average	**0.00017**	1.60	1.28	2.88	0.00073	1.43	1.11	2.54	0.00065	1.32	1.01	**2.32**

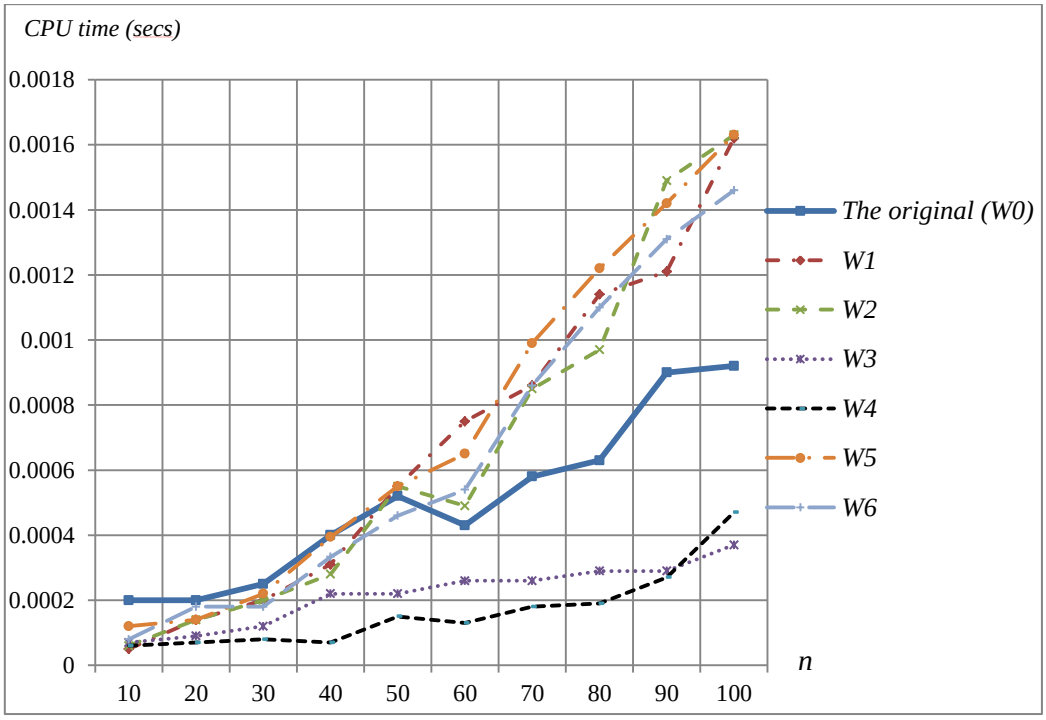


Figure 2.16: Average CPU time over 100 instances of the Enhancements for n = 10 to 100 (case of weighted)

In some cases, it has been observed that the differences in the CPU time between the original algorithm and other enhancements were very tiny. To highlight these differences, the deviation of the enhancements from the original algorithm was computed, see Table 2.5. The results were also given in Figure 2.17 for illustration purposes.

Table 2.5: Deviation (%) from the original (weighted, from n =10 to 100)

n	The Original (*W0*) Average CPU Time	Variant (*W1*)	Variant (*W2*)	Variant (*W3*)	Variant (*W4*)	Variant (*W5*)	Variant (*W6*)
10	0.00020	-75	-70	-65	-70	-40	-60
20	0.00020	-30	-30	-55	-65	-30	-10
30	0.00025	-20	-20	-52	-68	-12	-28
40	0.00040	-22.50	-30	-45	-82.50	-1.52	-16.67
50	0.00052	5.77	5.77	-57.69	-71.15	5.77	-11.54
60	0.00043	74.42	13.95	-39.54	-69.77	51.16	25.58
70	0.00058	48.28	46.55	-55.17	-68.97	70.69	48.28
80	0.00063	80.95	53.97	-53.97	-69.84	93.65	74.60
90	0.00090	34.44	65.56	-67.78	-70	57.78	45.56
100	0.00092	76.09	77.17	-59.78	-48.91	77.17	58.70
Average	0.000503	17.25	11.30	**-55.09**	**-68.41**	27.27	12.65

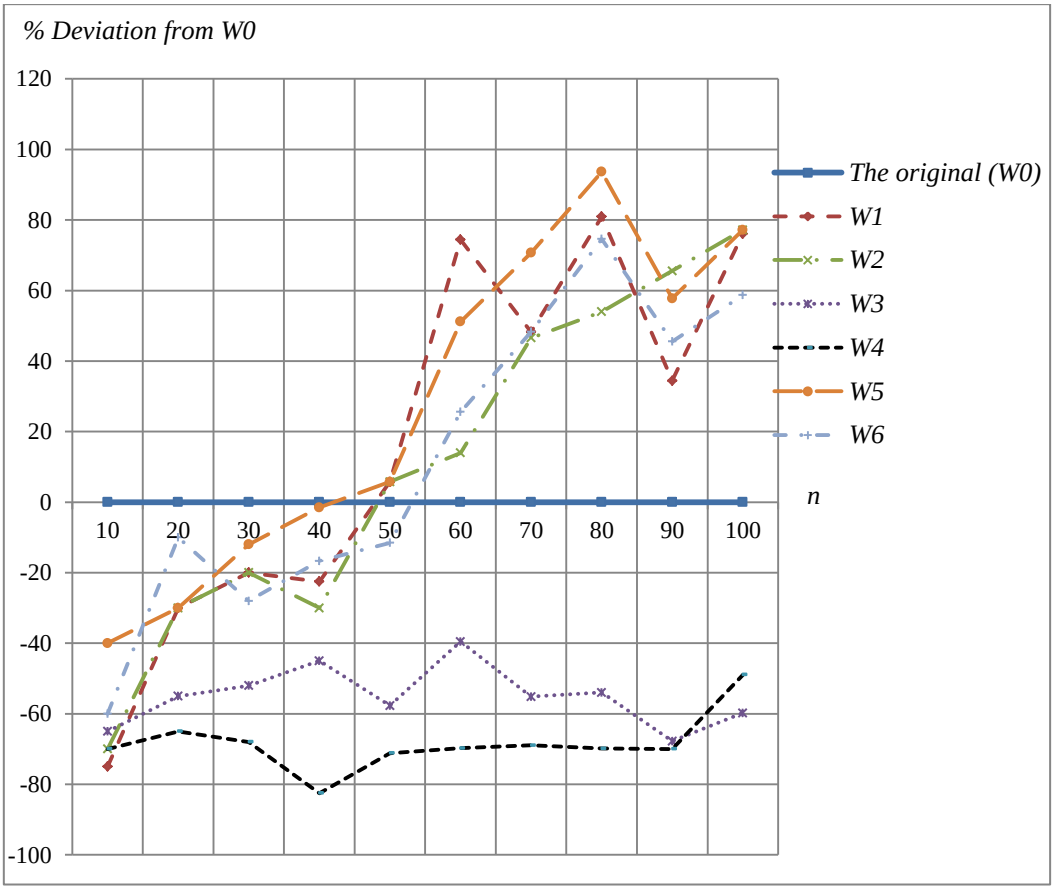


Figure 2.17: Deviation (%) of CPU time from the original algorithm (the weighted case, *n* from 10 to 100)

It can be noted that the use of the best enhancements (*W3* and *W4*) could save up to 68% and 83% of the time of the original algorithm when n = 90 and 40 respectively, with an average saving of about 55% and 68% respectively.

Additional tests for *W0*, *W3* and *W4*:

Since the differences in the CPU time between *W*3 and *W*4 were rather small especially when n =100 (0.00037 vs 0.00047 secs) as shown in Table 2.4, in this section, extra data sets were then used to further assess these two enhancements (*W*3 and *W*4), especially when the values of n are large. To be consistent with our previous experiments for the weighted case, we also used n = 50 to 1000 with an increment of 50 and using 100 instances for each value of n.

Table 2.6 provides the same information as in Table 2.4, where the average CPU for the original algorithm and our two enhancements were displayed in Figure 2.15.

Table 2.6: Average of CPU time and number of iterations over 100 instances for *W0*, *W3* and *W4* for n = 50 to 1000 (case of weighted)

n	*W0*				*W3*				*W4*			
	Average CPU	Iterations of			Average CPU	Iterations of			Average CPU	Iterations of		
		2 pt	3 pt	Total		2 pt	3 pt	Total		2 pt	3 pt	Total
50	0.00044	4.37	4.09	8.46	0.00017	1.70	1.44	3.14	0.00017	1.52	1.26	2.78
100	0.00070	4.71	4.35	9.06	0.00041	2.08	1.72	3.80	0.00015	1.71	1.36	3.07
150	0.00140	5.43	5.01	10.44	0.00051	2.18	1.77	3.95	0.00031	1.65	1.25	2.89
200	0.00177	5.63	5.24	10.87	0.00074	2.31	1.91	4.22	0.00031	1.75	1.35	3.10
250	0.00180	6.08	5.67	11.75	0.00096	2.50	2.09	4.59	0.00049	1.75	1.37	3.12
300	0.00210	5.74	5.37	11.11	0.00108	2.51	2.15	4.66	0.00065	1.74	1.39	3.13
350	0.00300	6.27	5.78	12.05	0.00131	2.50	2.03	4.53	0.00066	1.74	1.29	3.03
400	0.00320	6.63	6.12	12.75	0.00153	2.54	2.04	4.58	0.00077	1.83	1.36	3.19
450	0.00420	6.43	5.95	12.38	0.00142	2.60	2.12	4.72	0.00083	1.79	1.37	3.16
500	0.00392	6.52	6.04	12.56	0.00160	2.64	2.16	4.80	0.00083	1.80	1.38	3.18
550	0.00412	6.94	6.51	13.45	0.00158	2.68	2.25	4.93	0.00089	1.68	1.26	2.94
600	0.00465	6.87	6.47	13.34	0.00230	2.66	2.26	4.92	0.00110	1.64	1.26	2.90
650	0.00531	7.02	6.62	13.64	0.00257	2.65	2.25	4.90	0.00108	1.61	1.23	2.84
700	0.00571	6.89	6.46	13.35	0.00270	2.71	2.28	4.99	0.00112	0.53	1.47	2
750	0.00615	7.02	7.02	14.04	0.00270	2.76	2.33	5.09	0.00120	1.63	1.24	2.87
800	0.00637	7.26	6.83	14.09	0.00333	2.71	2.29	5	0.00121	1.61	1.20	2.79
850	0.00694	7.38	6.94	14.32	0.00360	2.85	2.42	5.27	0.00144	1.62	1.24	2.86
900	0.00724	6.91	6.40	13.31	0.00355	2.93	2.42	5.35	0.00152	1.76	1.29	3.05
950	0.00810	7.43	6.88	14.31	0.0039	2.84	2.29	5.11	0.00147	1.72	1.27	2.99
1000	0.00887	7.59	7.06	14.65	0.0042	2.86	2.34	5.20	0.00154	1.71	1.28	2.99
Average	0.00430	6.46	6.04	12.50	0.00201	2.56	2.13	4.69	0.00089	1.64	1.31	2.94

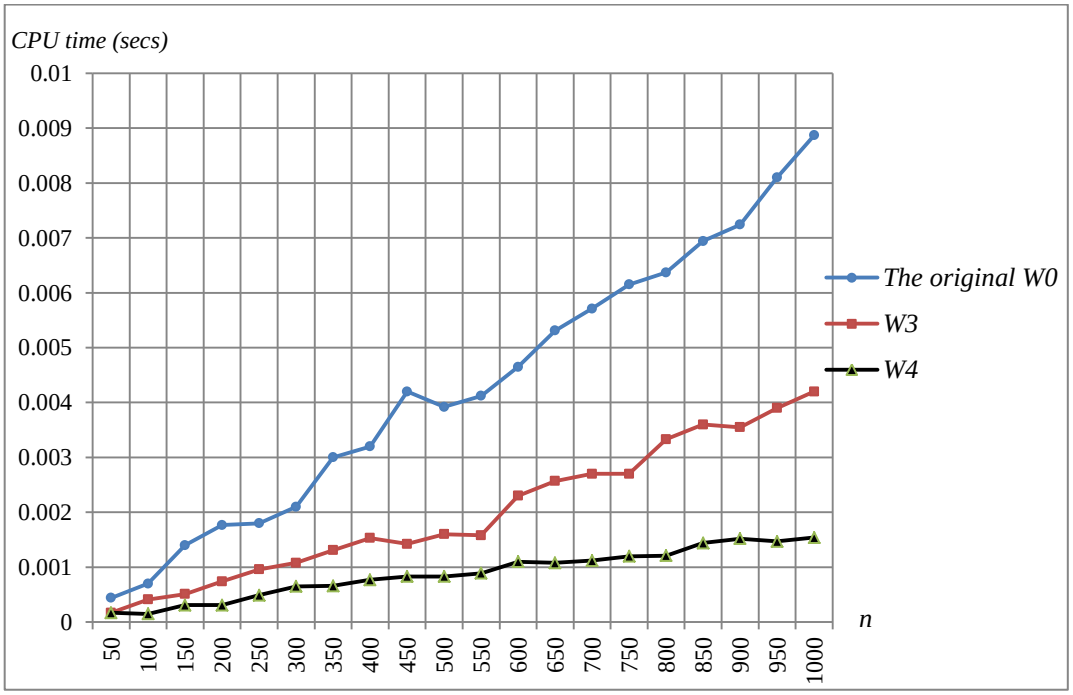


Figure 2.18: Average CPU time of *W0*, *W3* and *W4* (weighted, from n =50 to 1000)

Figure 2.18 showed that both enhancements (*W3* and *W4*) were found to be always much quicker than the original one, especially when n was large. It is worth mentioning that the results that were found in this section (when n =50 to 1000) were similar to the ones found in Table 2.4 (when n =10 to 100). However, the fourth variant (*W4*) was found to be consistently quicker than *W3* for all $n \geq 100$. Average CPU times (in seconds) of 0.00430, 0.00201and 0.00089 secs were reported for *W0*, *W3* and *W4* respectively.

In brief, on the basis of what we have mentioned above, we can conclude that *W4* was the best enhancement for solving the *1*-centre problem when incorporated as part of the *p*-centre problem for the weighted case. Moreover, the CPU time for the original algorithm was found to increase even more rapidly with n when compared to *W4*.

In general, we can confirm, based on our empirical results, that using our enhancements for solving the *1*-centre problem in both the unweighted and the weighted cases, namely (*V3*, *V4*) and (*W4*) respectively, yielded better results in terms CPU times than the other ones. The effect of these enhancements will be noted even more when these are used as part of local search for solving the *p*-centre problem ($p > 1$). This will be shown in the next section.

2.7 Effect of the enhancements on the continuous *p*-centre problem

In this section, a Multi-Start Alternate Locate Allocate algorithm is used to show the impact of the proposed enhancements when solving the *p*-centre problem ($p > 1$). This algorithm is used for the unweighted and the weighted case, where both the original Elzinga-Hearn algorithm and its enhanced versions are used as our local search.

The objective of the *p*-centre problem is to minimise the maximum distance between a demand point and its nearest facility.

The formulation of this problem is given earlier in chapter 1, see subsection 1.4.3.

As an example, consider in Figure 2.19 (*a*), where we have four possible locations (4 open facilities) for the *4*-centre problem. The objective function (minmax Z) for this case is the radius of the largest circle (circle centred at p_2). Figure 2.19 (*b*) shows a better solution for the same problem as the solution is improved ($Z' < Z$). In general, we cannot obviously state that this new solution is the optimal solution. However, we can confirm that the solution within each

region (the *1*-centre location problem) is the individual optimal solution. The *p*-centre is non convex and hence many local minima may exist. The problem is to find the right *p* clusters, which is in itself a hard combinatorial problem to solve. For each cluster the optimal solution can easily be found using Elzinga-Hearn algorithm or any of those chosen enhancements described in the previous section.

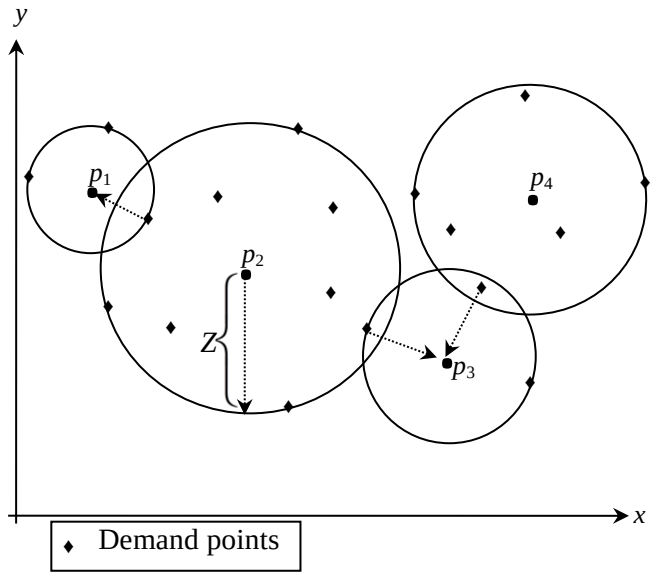


Figure 2.19 (*a*): A feasible solution of a *4*-centre location problem

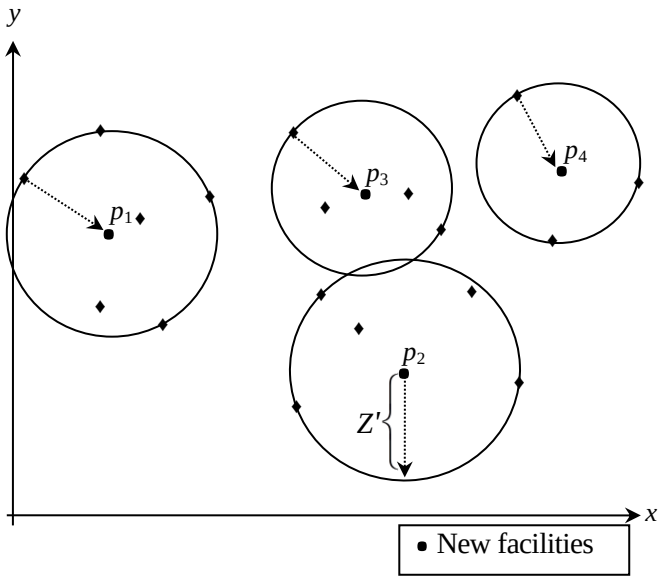


Figure 2.19 (*b*): A better solution of the same *4*-centre location problem

2.7.1 A random multi start alternate algorithm

The idea of this algorithm is to choose *p* initial facility points randomly and then divide the demand points set into *p* subsets. Each demand point is then allocated to the nearest facility. The exact location method (Elzinga-Hearn or proposed enhancements) is applied in each cluster to find the optimal single facility location. Each demand point is then reallocated again to the nearest facility. After all demand points have been completely reallocated, the exact location method is applied once again to improve the location of each facility. This procedure, which alternates between the location and the allocation phases, is repeated until no further improvement can be found. In order to have a better local minimum, this algorithm is repeated several times using different starting locations either randomly or using some forms of guidance that need to be defined. The main idea behind this algorithm is that one of them may lead to the right region which includes the global minimum. This principle is similar to the

alternate locate allocate procedure of Cooper (1964) used for the multi-source Weber problem. This is also similar to the heuristic described by Drezner (1984*b*) which is known as H1. Here, the original algorithm for both the unweighted and the weighted case, *V4* and *W4* will be used as part of our locate-allocate heuristic to find the optimal single facility minmax location problem in the continuous space in each region.

The main steps of this multi-start alternate locate-allocate approach, which we call MSALA for short, are given in Figure 2.20.

Step 1**:** Randomly select *p* initial starting locations. Allocate each demand point to the nearest facility to find *p* sets of demand point allocations.

Step 2: For each of the *p* independent set of allocations, solve optimally the *p* independent single facility location problem using the *1*-center problem on the continuous space (Elzinga-Hearn algorithm, variant *V4* or variant *W4*).

Step 3: Reallocate each demand point to its nearest facility for the new set of facility locations found in *Step* 2. If there are changes in the allocation, go to *Step* 2, otherwise stop and a local minimum has been found.

Step 4: Repeat *Steps* 1, 2, and 3 several times, say *K* times, and select the configuration with the least minmax distance.

Figure 2.20: Multi-Start Alternate Locate-Allocate Algorithm (MSALA)

In *Step* 1, *p* points are chosen randomly as initial starting points, however, this scheme could even be guided by introducing forbidden regions as explored by Luis *et al*. (2011) for the multi-source Weber problem. The implementation of *Step* 3 could also be made by recording the regions affected and devising a scheme for identifying neighbouring regions using Voronoi Diagrams as an example.

2.7.2 Impact of our enhancements (*V4* and *W4*) within MSALA (*Step* 2)

In this section we implement MSALA that uses the original algorithm and the enhancements (*V4* and *W4*) in *Step* 2 when solving the *1*-centre problem as part of the *p*-centre problem. To assess the performance of these enhancements, we use our MSALA on an instance with n=500 with several values of p (p = 3, 5, 10, 15 and 20). Here, 10 instances are generated for each case and average results are reported to show the impact of the

enhancement when using within the MSALA. We test this for both the weighted and the unweighted cases.

a) Computational results for the unweighted case

In this section, we present computational results of the MSALA for the unweighted case. Table 2.7 showed the average objective function values, the average CPU time for the original algorithm and the average number of times when the best solution was found. Here, 50 runs for the MSALA using the original algorithm (*V0*) were performed. Here, its CPU time was recorded which was then used as the stopping condition for the enhanced version (*V4*). Since this enhancement was much quicker than the original algorithm, the MSALA that used *V4* obviously performed more iterations than the one with *V0* as shown in Table 2.7.

For example, for p=10 the average CPU time (in seconds) for each instance with 50 runs was 3.23 with an average objective function value (Z) of 20.839. Using the same time, enhancement (*V4*) generated around 86.50 iterations with an average objective function values 20.513. The details were provided in Appendix *B1*, which consists of five tables (when n = 500 and p = 3, 5, 10, 15 and 20) where each table showed value of Z of MSALA for 10 instances when using *V0* and *V4*. In addition we reported the number of times when the best solution was found for both *V0* and *V4*, as well as the CPU time used for MSALA when using *V0* with 50 runs. For information, we also reported the total number of iterations required for *V4* when using the latter CPU time consumed by *V0* as our stopping criterion.

Table 2.7: Average results over 10 instances of MSALA for *V0* and *V4* (the unweighted case, with n = 500)

n=500	The Original algorithm (*V0*) (*for 50 runs*)			Variant (*V4*) (*Using the total CPU time of V0*)		
p	Average Z	Average CPU Time (secs)	Average number of bests*	Average Z	Average total # iterations+	Average number of bests*
3	47.600	1.243	4.3	47.600	88.6	6.8
5	30.031	1.867	1.1	30.006	92.8	1.8
10	20.839	3.227	0.5	20.513	86.5	1
15	17.270	4.062	0.7	17.187	61.4	1
20	14.841	5.015	0.7	14.797	60.8	1
Average	26.116	3.083	1.5	**26.021**	78	**2.3**

*: *Number of times when the best solution was found divided by10 (instances).*

+: *Total number of iterations when using the CPU time of V0 (50 runs) divided by10 (instances).*

In brief, we can say that in most cases the values of the objective function when using the enhancement (*V4*) was slightly better than those found by the original algorithm. It has been observed that *V4* performed 56% more iterations than *V0* while consuming the same CPU time (about 78 iterations instead of 50). For smaller values of p ($p \leq 10$) the number of iterations used was even larger. Figures 2.21 and 2.22 showed a summary of the comparison between *V0* and *V4*, in terms of the total number of iterations and the average number of iterations for finding the best solutions respectively.

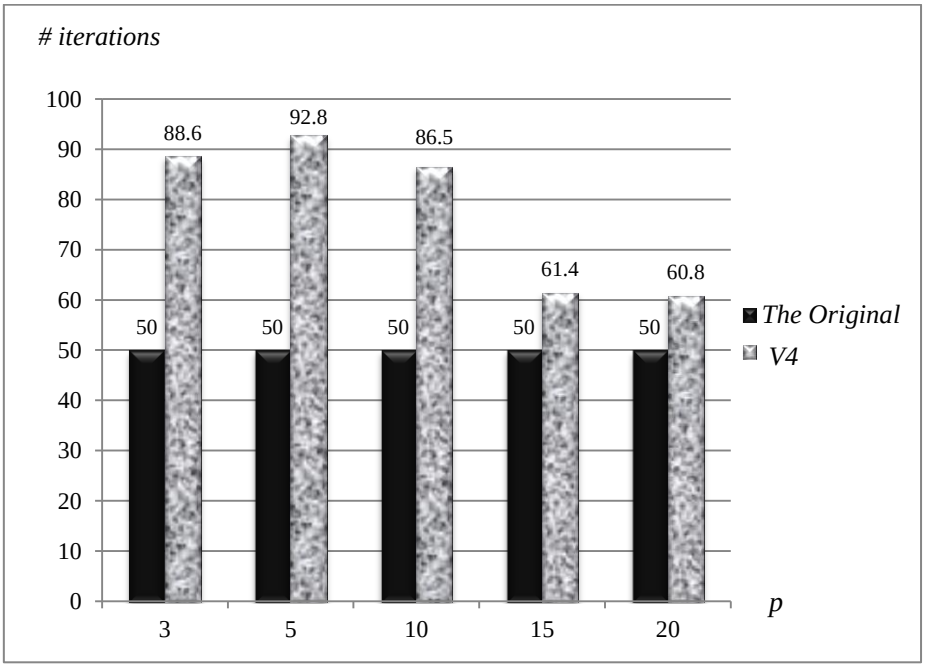


Figure 2.21: The average number of iterations for the MSALA using *V0* and *V4* for the unweighted case (n=500) based on 50 runs of MSALA

As we mentioned above, in most cases the values of the objective function when using *V4* was slightly better than those found by *V0*. On the other hand, we can note that there was a clear difference in terms of the number of iterations used between *V0* and *V4* with a significant deviation of up to 92.8%. This significant difference in the number of iterations provided a greater opportunity for our heuristic (or any other meta-heuristic when used in the *p*-centre problem) to get a better solution without requiring extra CPU time.

However, there was in general no significant difference between the values of the objective function of *V0* and *V4*. The reason for this was due to the performance of the MSALA algorithm which is relatively poor compared to the other meta-heuristics, as will be shown in subsequent chapters. In other words, this approach is considered to be a blind search as a

significant number of additional iterations was used without necessary improving the solution.

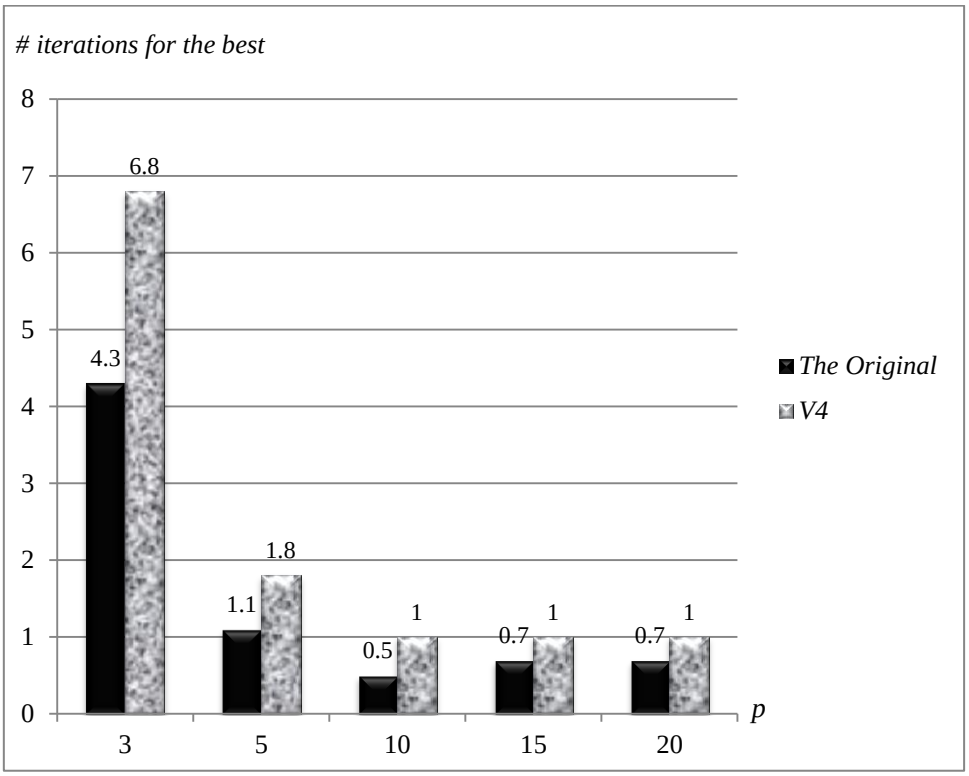


Figure 2.22: The average number of iterations when the best solution was found for the MSALA using *V0* and *V4* (*n*=500)

b) Computational results for the weighted case

In this section, we will provide the same information as presented in Table 2.7, where the results of the original algorithm for the weighted case (*W0*) versus those of *W*4 were compared. The results were also summarised in Table 2.8 whereas the detailed results can be found in Appendix *B2*.

In brief, similar results, as found in the unweighted case, were also observed here. A summary of the comparison between *W0* and *W*4, in terms of the total number iterations, was displayed in Figure 2.23, where the overall average value of *W4* is 77.8 vs 50 iterations for *W0*. The average number of iterations when the best solution was found, was displayed in Figure 2.24, where their overall average values were 2.9 and 1.6 for *W0* and *W4* respectively.

Table 2.8: Average results over 10 instances of MSALA for *W0* and *W4* for n = 500 (the weighted case)

n=500	The Original algorithm (*W0*) (*for 50 runs*)			Variant (*W4*) (*Using the total CPU time of W0*)		
p	Average *Z*	Average CPU Time (secs)	Average number of bests*	Average *Z*	Average total # iterations+	Average number of bests*
3	406.218	1.211	3.9	405.797	90.8	8.2
5	258.785	1.878	1.9	258.785	85.5	2.7
10	178.981	2.833	0.7	178.072	80.3	1.1
15	146.069	3.143	0.8	144.917	68.6	1.2
20	126.007	3.588	0.7	124.019	63.6	1.1
Average	223.212	2.531	1.6	**222.318**	77.8	**2.9**

*: *Number of times when the best solution was found divided by10 (instances).*

+: *Total number of iterations when using the CPU time of W0 (50 runs) divided by10 (instances).*

In the weighted case, we have observed similar results as found in the unweighted case. A summary of the comparison between *W0* and *W*4, in terms of the total number iterations, was displayed in Figure 2.23. The overall average value of *W4* was 77.8 vs the 50 iterations for *W0* showing nearly a 55% increase in the number of iterations while using the same CPU time of *W0*. In terms of the number of iterations when the best solution was found, this was displayed in Figure 2.24, where their overall average values were found to be 2.9 and 1.6 respectively.

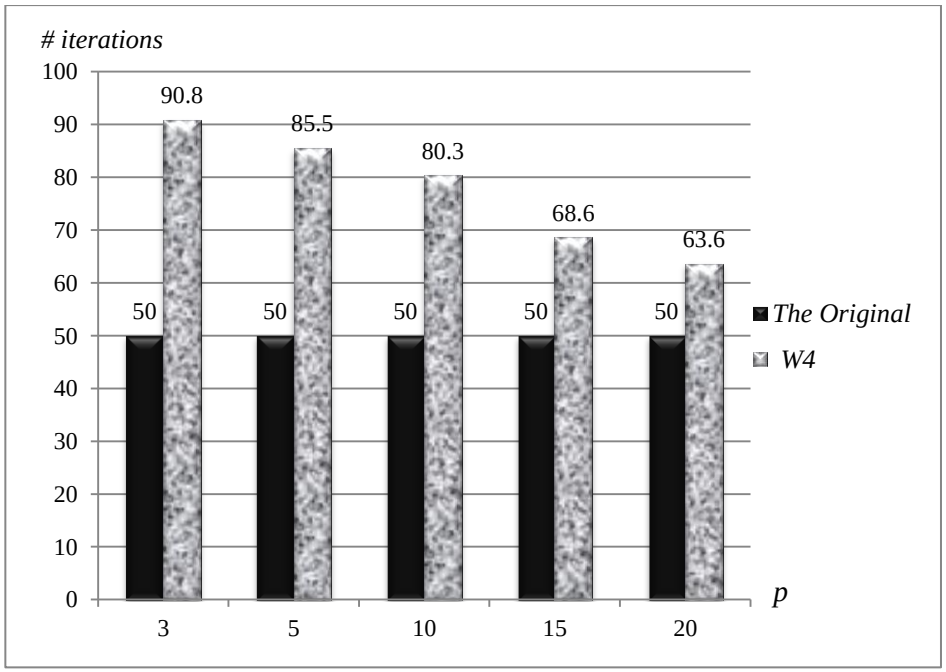


Figure 2.23: The number of iterations for MSALA using *W0* and *W*4 for the weighted case (n=500)

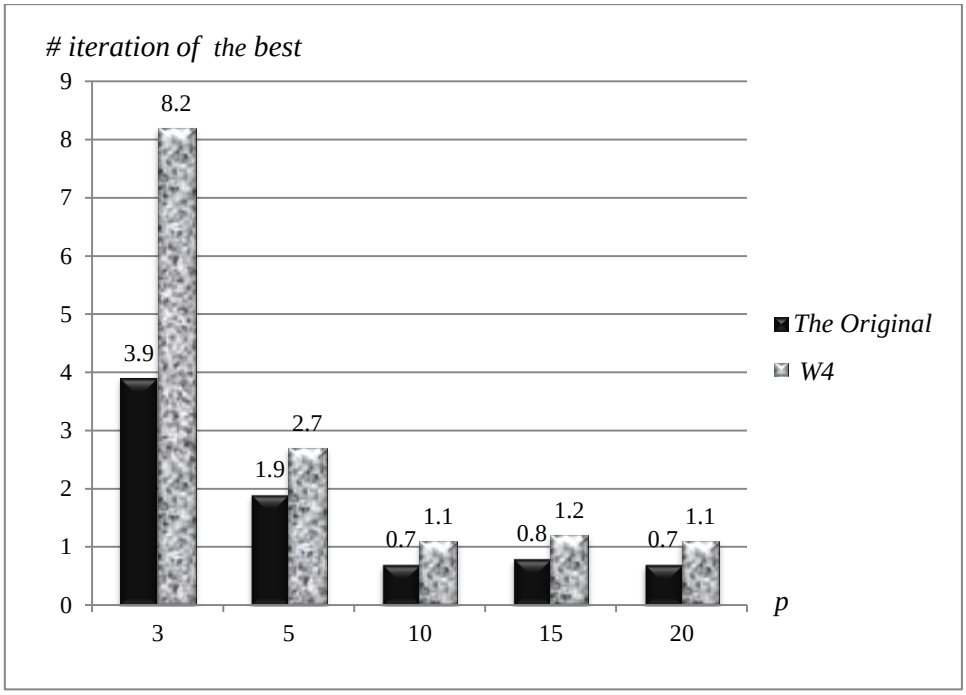


Figure 2.24: The average number of iterations when the best solution was found for the MSALA using *W0* and *W*4 for the weighted case (*n*=500)

As we mentioned above, although there was a significant difference between the number of iterations used and the number of iterations when the best was found when using *W0* and *W*4, in this particular set of instances, there was however not a significant difference between their objective function values.

2.8 Summary

In this chapter, we presented a review of the Elzinga-Hearn algorithm aimed at solving optimally the *1*-centre location problem in the continuous space. Furthermore, we proposed six enhancements for the unweighted and another similar six for the weighted case to the original implementation of the well-known Elzinga-Hearn algorithm. Our enhancements (*V3* and *V4* for the unweighted case and variant *W3* and *W*4 for the weighted case) outperformed the original implementation of the algorithm for all values of *n* which we tested. In brief *V3* and *V4* yielded about 56% and 64% reduction in CPU time over *V0* respectively. Similarly *W3* and *W*4 produced about 55% and 68% time reduction over *W0* respectively. To distinguish between the performance of the two chosen enhancements, extra tests using large value of *n* were performing which demonstrated that *V3* (when $n > 70$) and *V4* (when $n \leq 70$)

for the unweighted and *W4* for the weighted case are the best performers. To assess the performance of our enhancements for solving the *p*-centre problem, the Elzinga-Hearn algorithm and our enhancements (*V4* and *W*4) were used as local search within a multi-start based approach based on the Cooper's 'locate-allocate' principle. Ten instances of size n=500 with various values of p were used. Our enhancements were found in contribute to producing slightly better results when compared to using *V0* and *W0* instead.

In the next chapter a powerful meta-heuristic, namely a Variable Neighbourhood Search (VNS) is explored to investigate the problem further.

Chapter 3

Adaptive VNS-Based Heuristics

3.1 Introduction

In this chapter, a brief review of Variable Neighbourhood Search (VNS) is first given. Two neighbourhood structures namely the Customer-based Neighbourhood (*CN*) and the Facility-based Neighbourhood (*FN*) are explored followed by powerful enhancements. In addition, effective refinements on the local search are also proposed. A learning scheme is then embedded into the search to produce an adaptive VNS with memory. These variants are tested on several large instances with encouraging results.

The basic VNS

The basic idea of VNS is to change neighbourhoods systematically while using a local search within each neighbourhood to get to a corresponding local minimum. A brief outline of the basic VNS approach is given in Mladenovic and Hansen (1997) but new versions as well as advanced implementations and applications can be found in Hansen *et al.* (2010). The different neighbourhood structures which we constructed are originally based on those used for the multi-source Weber problem with some additional changes to cater for the properties of the minimax objective function. In other words, we use the Elzinga-Hearn algorithm, or its enhanced versions, to solve the *1*-centre problem instead of a Weiszfeld algorithm that is used for solving the *1*-median problem. These include customer-based moves (e.g., the removal/addition of one or more demand points from a region), and facility-based ones (e.g., opening/closing one or more facilities). This search continues until all the neighbourhoods have been searched, the allowed time (the total CPU_{max}) is reached or any other stopping rule (the total number of iterations or the number of iterations between two successive improvements) is met. The steps of the basic VNS algorithm is given in Figure 3.1 with some of the main steps explained next.

Step 0: Find an initial solution X, define the sequence of neighbourhoods N_k, k=1,…, K_{max}; choose a stopping condition.

Step 1: Repeat the following steps until the stopping condition is met:

(a) Set k = 1.

(b) Repeat the following steps until $k = K_{max}$:

(i) Generate a neighbouring solution X' at random from the k^{th} neighbourhood of X ($X' \in N_k(X)$). ***"Shaking Step"***

(ii) Apply a local search based on N_k using X' as the initial solution to obtain a local optimum X''. ***"Local Search Step"***

(iii) If $Z(X'') < Z(X)$, set $X = X''$, and k=1; (*Move or not*)
otherwise, set $k = k$+1;

Figure 3.1: Steps of the basic Variable Neighbourhood Search (VNS)

Neighbourhood Structures

***Step* 1b (i): (shaking)**

In this study, we use two types of neighbourhood, each with a maximum size K_{max} which we refer to for simplicity as Max_1 and Max_2 respectively. These include:

a) CN: Customer-based Neighbourhood

We remove k customers randomly from their assigned facilities and allocate them randomly to the open facilities. This neighbourhood $N_k(X) = CN_k, k = 1,..., K_{Max_1}$ will be defined in subsection 3.2.

b) FN: Facility-based Neighbourhood

This type of neighbourhood is divided into two subtypes as follows:

We remove k open facilities randomly and replace them with k facilities randomly selected

(i) at demand points (customers sites) or

(ii) anywhere in the plane.

This neighbourhood which we denote by $N_k(X) = FN_k$, $k = 1,..., K_{Max_2}$ will be defined in subsection 3.3.1.

***Step* 1b (ii): (Local search)**

This step is made up of two phases:

- Improve the location within each cluster using the Elzinga-Hearn algorithm or the enhancements proposed in Chapter 2.
- Perform the locate-allocate procedure, which will be described in subsection 3.2.1.

The next two sections cover the customer-based and the facility-based VNS respectively.

3.2 The customer-based VNS(*CN*)

In this section, we will first apply the original algorithm for the customer-based neighbourhood (*CN*) and then introduce three enhancements, followed by some computational results.

The idea is to choose a number of demand points (*k* demand points, $k=1,..,K_{\max}$), which are then allocated to other clusters (other facilities). The locate-allocate procedure of Cooper's approach is then introduced as a local search to improve upon the solution. If a better solution is found, we record such a solution (X) and start the procedure again with $k=1$ (i.e., $CN_1(X)$); otherwise the next neighbourhood $CN_2(X)$ is explored. Here, two demand points, ($k=2$) are reallocated to other facilities. This search continues until $K_{\max}$ is reached where we revert back to $k=1$.

The construction of $CN_k(X), k=1,...,K_{\text{Max}_1}$ is based on the following definitions:

Let $W_j = \{P_i \in I : d(P_i, X_j) = \underset{r=1,...,p}{\text{Min}}\, d(P_i, X_r)\};\quad j=1,...,p$ (3.1)

be the set of customers served by facility F_j (i.e., facility F_j located at X_j).

Let $B_k \subset I$ be the subset of customers from I with $|B_k| = k$ that are selected for removal (randomly or using other means as we will see later) (3.2)

$D_j^-(B_k) = B_k \cap W_j$ be the set of customers from W_j that are selected to be removed; $j=1,...,p$

$D_j^+(B_k) = \{P_i \in B_k : P_i \text{ is allocated to facility } F_j\}$ be the set of customers from B_k that are randomly reallocated to facility F_j; $j=1,...,p$

Let $\hat{W}_j(B_k) = W_j \setminus D_j^- \bigcup D_j^+$ be the set of customers that are assigned to facility F_j (3.3)

Define $X'_j(B_k)$: centre of the smallest circle that encompasses all customers (points) in

$\hat{W}_j(B_k)$ (3.4)

Note that $X'_j(B_k)$ can be found by Elzinga-Hearn or other enhanced methods given in chapter 2.

Let $CN_k = N_k(X) = \{X'_1(B_k), ..., X'_p(B_k)\}$ (3.5)

Figure 3.2 illustrates the above idea where Figure 3.2 (*a*) shows a feasible solution of a *4*-centre location problem with centres located at p_1, p_2, p_3 and p_4. The allocation of the demand point (A) to the facility centred at p_2 shows a new feasible and better solution after the local search is used, see Figure 3.2 (*b*). Here, in the allocation process, the facility location of the source cluster is moved from p_1 to $\bar{p}_1$ and the new facility location of the destination cluster is moved from p_2 to $\bar{p}_2$. In this particular example, the other clusters were not affected though this not always true.

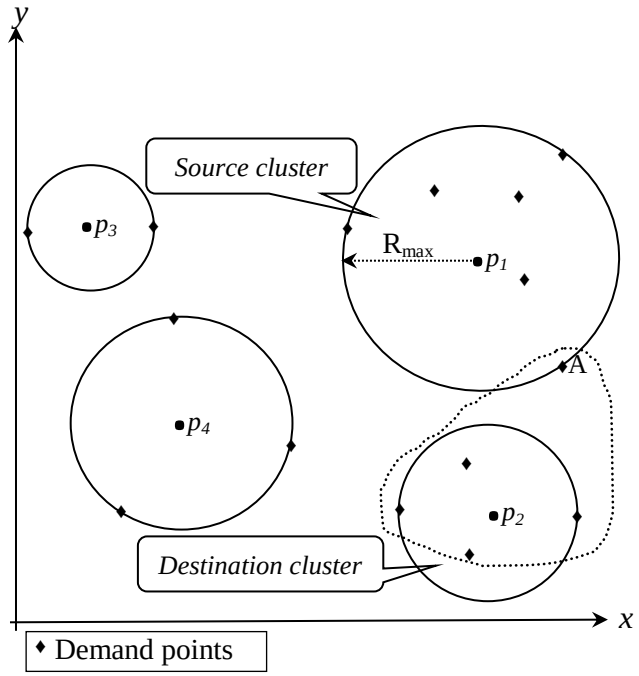


Figure 3.2 (*a*): A feasible solution of a *4*-centre location problem

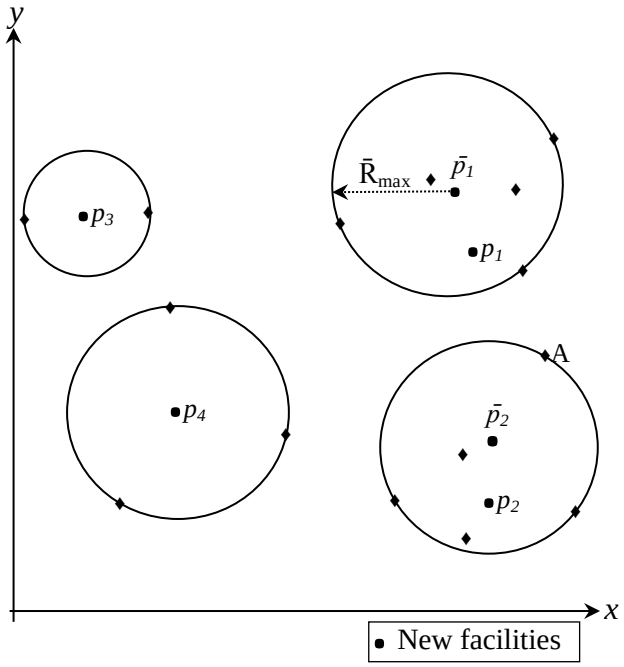


Figure 3.2 (*b*): A better solution of the same *4*-centre location problem

It is worth noting that it is not always possible that the allocation process yields a better solution. In order to test the efficiency of this allocation scheme, we will first apply this original algorithm and then explore three enhancements.

3.2.1 The original VNS algorithm using Customer-based Neighbourhood (*CN*)

This algorithm, which we call VNS(*CN*), is summarised in Figure 3.3 and a brief explanation of the main steps is given here.

Step 0: Specify $K_{\max}$ and $\text{CPU}_{\max}$ and set $Time = 0$. Define the sequence of the neighbourhood structures CN_k; k=1,…,$K_{\max}$ using (3.1)−(3.5).

Step 1: Generate an initial feasible solution (X), record the objective function $Z(X)$ and set $k = 1$.

Step 2:

- *Step 2a:* Generate a neighbouring solution $X' \in CN_k(X)$ using (3.1)−(3.5) ***"Shaking Step"***
- *Step* 2*b*: Apply a local search to obtain X'' starting from X' ***"Local Search Step"***
- *Step* 2*c*: If $Z(X'') < Z(X)$ set $X = X''$ and k=1. ***"Evaluation Step"***

  Else

  if $k = K_{\max}$ set $k = 1$, else set $k = k + 1$.

Step 3: Stopping condition

If *Time* > $\text{CPU}_{\max}$ record the incumbent solution *X* and stop, else go to *Step* 2.

Figure 3.3: A basic Customer-Based VNS Algorithm (VNS(*CN*))

A brief explanation of the main steps

Step 0 (the construction of the neighbourhood structures)

Here we remove *k* customers randomly and allocate them to other open facilities. We refer to this type of neighbourhood as CN_k; k=1,…,$K_{\max}$ with $K_{\max}$ denoting the number of neighbourhood structures.

In the preliminary study, we observed that the performance of VNS is likely to be more efficient when the value of $K_{\max}$ is relatively small. In addition, we noted that this relates to the number of open facilities (i.e., *p*), and setting $K_{\max}$ to $\lceil \sqrt{p} \rceil$ is found to be the most promising choice.

Step 1 (the initial solution)

This is generated randomly by choosing *p* fixed points, though other schemes could also be used.

Step 2b (the local search)

A locate-allocate procedure, which is similar to that of Cooper (1964), is used here.

(i) Given the *p* locations with their centres $X_j\,(j=1,...,p)$, allocate each customer to its nearest centre and define for each centre X_j, the subset W_j, as defined in (3.1)

(ii) In each subset $W_j\,(j=1,...,p)$, determine the optimal location X_j using our enhancements on the Elzinga-Hearn algorithm (i.e., if $n \leq 70$ use *V4*, else use *V3*, as described in chapter 2).

(iii) While there is a change in at least one of the subset W_j or in the location $X_j; j=1,...,p$, return to (i), else record the incumbent solution *X* and stop.

This local search will be revisited in section 3.4.

3.2.2 Possible schemes in the construction of the set B_k (*Step* 0).

As we have mentioned earlier, any circle can be determined by three critical points on its circumference (edges of an acute triangle) or by two critical points on the two ends of its diameter. When the source cluster (source circle) contains a larger number of points than the number of its critical points, the selected point can be either a critical point or not. Therefore, the random selection of customers in B_k as defined by (3.2) can be classified into two main cases: In the first case, a non-critical point is chosen, whereas in the second case only a critical point can be selected. In the first case, there will be a change in the facility location of the destination cluster only. However, if a critical point is selected, there will be a change in the facility locations of both the source and the destination clusters. In the next subsections we will incorporate this information to modify the original algorithm VNS(*CN*) in order to improve its efficiency under subsections (*a*) and (*b*).

a) Initial construction of the set B_k

Here the customers in B_k are randomly chosen from I. These customers can be critical or non-critical. In this case, we can conclude that the solution cannot be improved before applying the local search. However, when the local search is used, the change that has occurred in the facility of the destination cluster may or may not improve the solution. In the next subsections, both cases are discussed.

i) There cannot be an improvement in the solution before applying the local search

When a non-critical point is selected and allocated to another cluster, its inclusion leads to an increase in the radius of the destination circle without reducing the radius of the source cluster. For instance, Figure 3.4 shows a feasible solution of four clusters whose facilities are located at p_1, p_2, p_3 and p_4. Also it shows allocating point a_1 of cluster centred at p_1 (source cluster) to the destination cluster centred at (p_2), resulting to the destination cluster (including point a_1) becoming larger with its new facility located at $\bar{p}_2$.

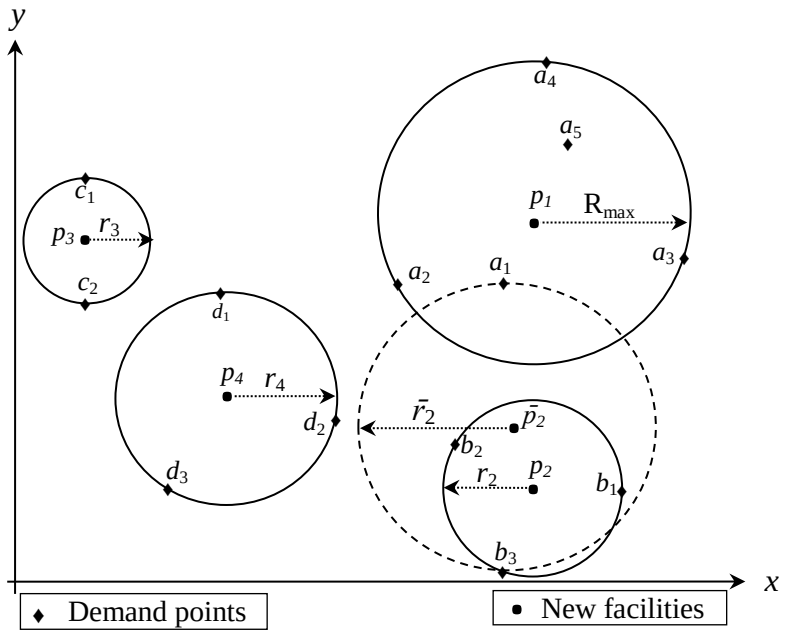


Figure 3.4: A feasible solution of a *4*-centre location problem using a non-critical point as neighbour before applying the local search

The source cluster when solved without point a_1 is not affected as its facility has not changed with its centre still at p_1. This is because point a_1 is not a critical point. From

Figure 3.4, we can also note that there has been no reduction in the radius of any circle. Therefore, the solution will not improve and can even yield to a worse solution if we only apply this phase.

ii) Effect of the local search

When the local search is applied, the change that has occurred in the facility location of the destination cluster can now lead to a change in the facility locations of some or all of the other clusters, which may produce a better solution. For instance, Figure 3.5 (*a*) shows a feasible solution of four clusters with facilities located at p_1, p_2, p_3 and p_4 and point a_1 is now allocated to the destination cluster with centre p_3. Thus the new facility of this cluster containing the point a_1 is now moved to $\bar{p}_3$, though the facility of the source cluster (cluster p_1 without point a_1) has not changed. However, when the local search is applied, the points (c_1, c_2, a_2 and d_1) are allocated to the destination cluster with a facility centred at $\bar{p}_3$, as these points are now closer to facility $\bar{p}_3$ than their initial facilities. In this particular example, we can say that the change that has occurred in the facility of the destination cluster has an effect in the reallocation of other points which in turn led to a better solution. From this, we can note that the change that occurred in the facility located at p_3 (moving from p_3 to $\bar{p}_3$) will lead to a change in the facility locations of the clusters with facilities centred at p_1, p_3 and p_4. This is because clusters with centres p_1 and p_4 has each one lost a critical point (a_2 and d_1), when the destination cluster with centre p_3 received these points, which contributed to a better solution.

We can therefore conclude that the solution can improve, if the change in one of the new facility locations attracts one of the critical points of the largest circle. In other words, one of the critical points is now closer to one of the facilities than its initial facility. For instance, in Figure 3.5 (*a*), the point (a_1) is chosen and reallocated to facility centred at p_3, therefore this facility (p_3) is moved to $\bar{p}_3$ and the critical point a_2 became closer to $\bar{p}_3$ than its initial facility (p_1). This could improve the solution when the local search is applied. However, this is not always the case**.** For example, in Figure 3.5 (*b*) though reallocating point a_5 to facility p_3 led to moving this facility to $\bar{\bar{p}}_3$. The critical point (a_2) is not allocated to facility $\bar{\bar{p}}_3$ even when the local search is used. This is because point (a_2) is still closer to its original facility (p_1) than the location of facility $\bar{\bar{p}}_3$. On the basis of what we have mentioned above, we can conclude that allocating a critical point to another cluster is likely

to be more efficient than allocating a non-critical point. This is explored in the next subsection.

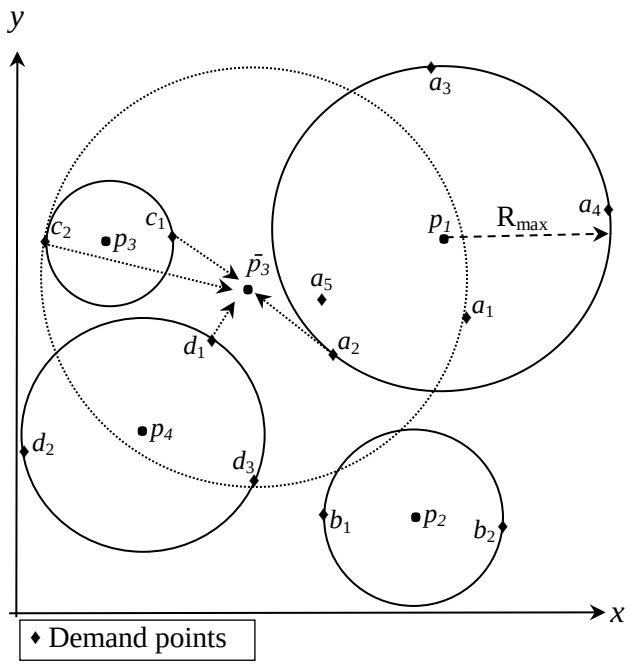


Figure 3.5 (*a*): Reallocating a non-critical point that can improve the solution when applying the locale search

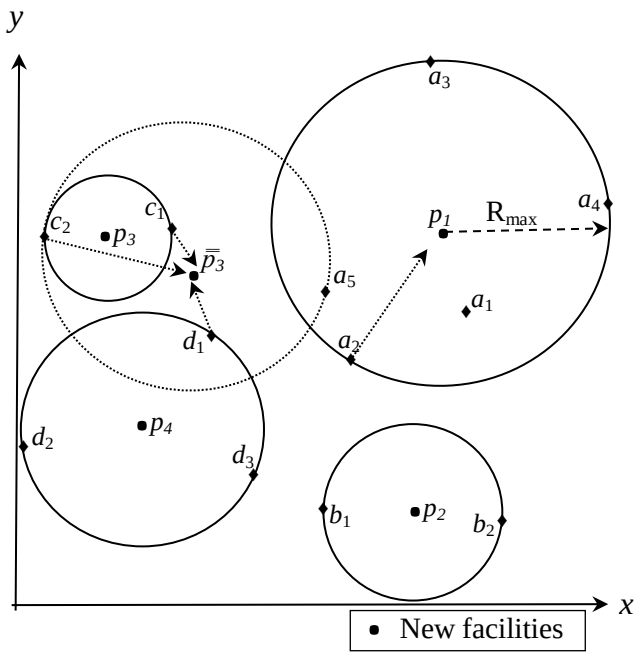


Figure 3.5 (*b*): Reallocating a non-critical point that cannot improve the solution when applying the locale search

b) The construction of the set B_k using critical points only

In this case, we only select critical points to make up the set B_k. When we choose a critical point to be allocated to another cluster, a change will occur in the facility locations of both destination and source clusters. Because the source cluster will lose at least one of its critical points, its radius could decrease. However, the destination cluster will receive at least one extra point, which originally was not assigned to it, and hence its radius could increase.

We can therefore claim that the solution can improve before applying the local search only if the following conditions are satisfied:

(i) *one of the critical points of the largest circle are allocated to another circle*

(ii) *the new radius of the destination cluster is still less than the old radius of the source cluster.*

However, even if (*ii*) is not achieved, the solution can be improved after the local search is applied. Therefore, we can classify the improvement in the solution into two cases, namely before or after applying the local search. These two scenarios are described next under (*i*) and (*ii*) respectively.

i) Possible improvement without the local search

In this section, we explain the improvement in the solution before applying the local search. Figure 3.6 (*a*) shows a feasible solution of four clusters with facilities located at p_1, p_2, p_3 and p_4 where a point a_1 is allocated to the other cluster (p_3). The source cluster (without point a_1) and the destination cluster (with point a_1) are then examined where the new facility location of the source cluster will be at $\bar{p}_1$ and p_3 will move to $\bar{p}_3$. It can be noted that the solution has improved before applying the local search, because a critical point of the largest circle is allocated to another cluster when the new radius of the destination circle (R_3) is found to be less than the old radius of the source circle namely R_{max}, see Figure 3.6 (*b*).

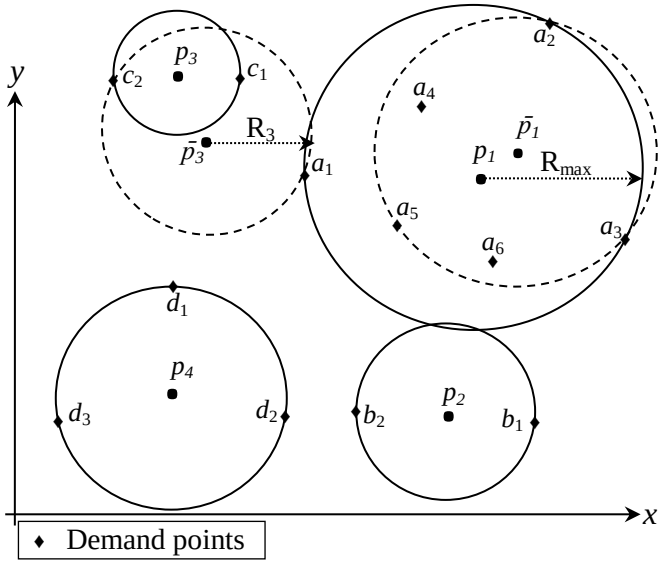


Figure 3.6 (*a*): A feasible solution of a *4*-centre location problem

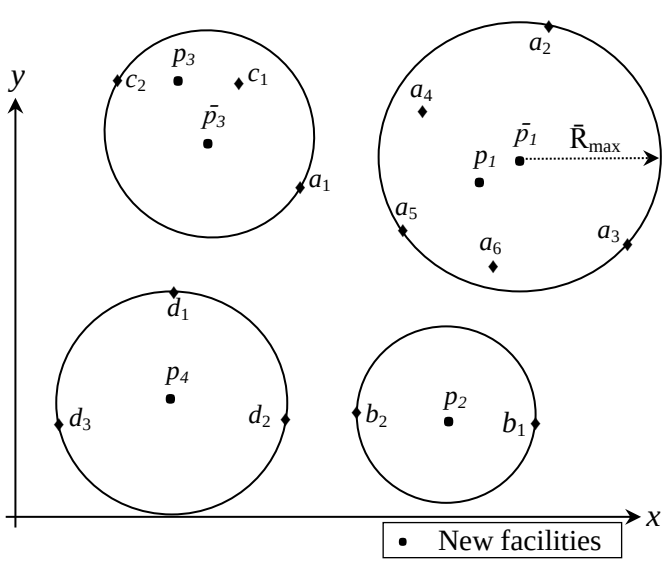


Figure 3.6 (*b*): Improvement, before applying the local search, because of a change in the source cluster

Note that this type of improvement is not possible if a non-critical point is chosen instead, because the source circle or any other circle will not get smaller, hence increasing the radius of the destination circle.

ii) Improvement after the local search

When a critical point is chosen, the facility location of both the source and the destination clusters will change. When the local search is applied, this change can lead to allocating one of the critical points of the largest circle to one of the other facilities, resulting in an improved solution. In other words, the improvement can happen either because of a change in the facility of the source cluster or a change in the facility of the destination cluster. This differs from the case of choosing a non-critical point where the solution can be improved due to a change in the facility of the destination cluster only. In the following, we will explain such an improvement due to a change in the source cluster. Figure 3.7 (*a*) shows a feasible solution of four clusters with facilities located at p_1, p_2, p_3 and p_4 where the point b_3 is allocated to cluster p_4. When the centre of the destination cluster is re-examined (cluster p_4 containing the point b_3) its facility has changed to $\bar{p}_4$. Similarly, when the source cluster is examined (cluster p_2 without point b_3), its new facility has now moved to $\bar{p}_2$.

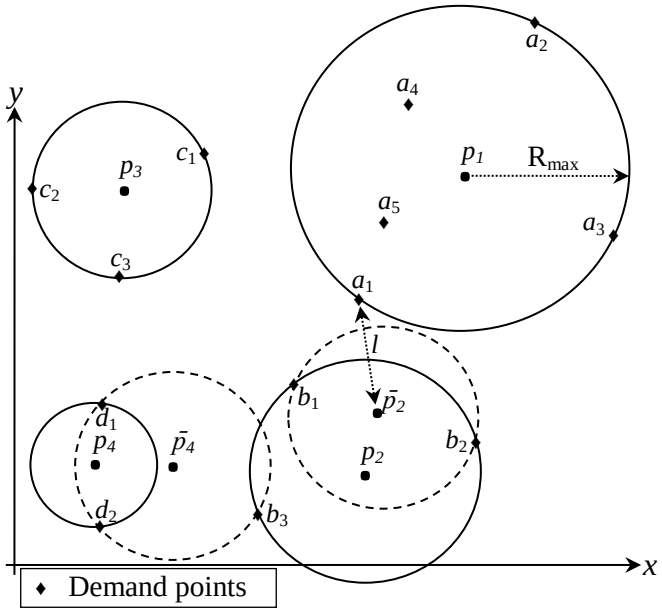


Figure 3.7 (*a*): A feasible solution of a *4*-centre location problem

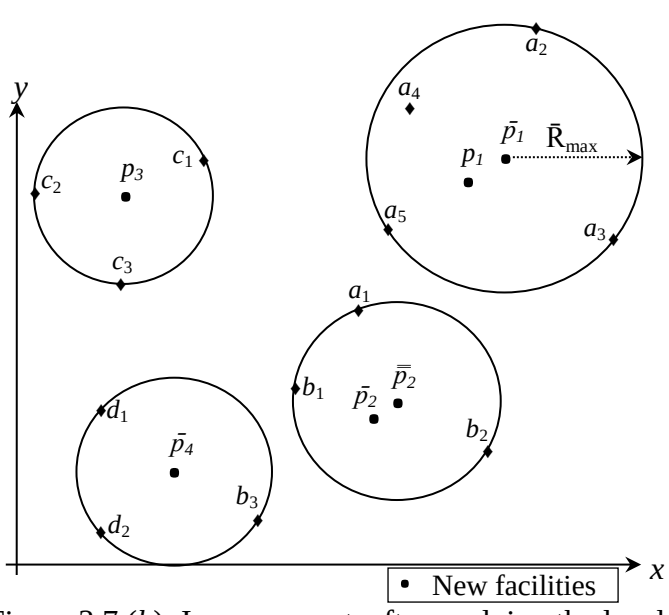


Figure 3.7 (*b*): Improvement, after applying the local search, because of a change in the source cluster

It can be noted that point a_1 became closer to $\bar{p}_2$ than its initial facility ($l < R_{max}$). This is because of the change in facility location of the source cluster. When the local search is applied, point a_1 is allocated to source cluster ($\bar{p}_2$), as shown in Figure 3.7 (*b*). Facility $\bar{p}_2$ is now moved to $\bar{\bar{p}}_2$ when its cluster includes point a_1 and p_1 is also moved to $\bar{p}_1$ (cluster p_1 without point a_1). This means that the solution will improve as $\bar{R}_{max} < R_{max}$.

In summary, it is worth claiming that the improvement in the solution occurred because of the change in the facility of the source cluster. Such an improvement cannot happen in the case of choosing a non-critical point, where the change occurs in the facility location of the destination cluster only.

In brief, on the basis of what we have mentioned, we can conclude that choosing a critical point is more efficient than choosing a non-critical point. In addition, allocating one of the critical points of the largest circle will have a positive impact on improving the solution, because the objective function aims to minimise the maximum distance. Note that R_{max} may not decrease if the solution has more than one largest circle. In this case the repeat of this scheme will alleviate this drawback.

Here, we propose three enhancements, in the first enhancement (*CNV1*), we will focus on allocating the critical points to other clusters randomly. The idea behind the second enhancement (*CNV2*) is to select a point from the largest circle and allocate it to another cluster, which is more efficient than choosing a point from another circle. We then combine both ideas to produce the third enhancement which we call (*CNV3*).

3.2.3 Variant (*CNV1*): Critical points-based neighbourhoods

Instead of choosing randomly any point (a critical or a non-critical point) to define B_k, the critical points from all the p circles will be selected only. A discussion about this construction is given in the previous subsection. The steps of this variant are the same as those of the original algorithm of Figure 3.3, except that *Step* 0 is replaced as follows:

Step 0: Specify CPU_{max} and set $Time = 0$. Define the sequence of neighbourhood structures NC_k using $B_k \subset E_C$ with $|B_k| = k$ instead of (3.2) with E_C denoting the set of critical points of all the circles. Note that (3.1) and (3.3 – 3.5) remain unchanged.

3.2.4 Variant (*CNV2*): Largest circle-based neighbourhoods

We can say that the random selection of a point from the largest circle and allocating it to another circle can reduce the radius of the largest circle. Such an improvement is not necessarily guaranteed as the destination circle with the addition of the new point, when solved, may lead to a greater radius than the previous radius of the source circle. However, in the case of choosing a point from the largest circle, the possibility of reducing the radius of

the largest circle will be higher than the possibility of the other case (choosing a point not from the largest circle) except in the case of ties. Therefore, we choose some or all of the points of the largest circle to determine B_k in (3.2) instead and we allocate them randomly to the other clusters. The main steps of this variant are the same as the ones of the original algorithm, except that *Step* 0 of Figure 3.3 is replaced by:

Step 0: Specify $\text{CPU}_{\max}$ and set $Time = 0$. Define the sequence of neighbourhood structures NC_k using $B_k \subset E_G$ with $|B_k| = k$ instead of (3.2) with E_G denoting the set of points encompassed by the largest circle. Note that (3.1) and (3.3 – 3.5) remain unchanged.

3.2.5 Variant (*CNV3*): Critical points of the largest circle-based neighbourhoods

In this enhancement, we will combine both variants (*CNV1*) and (*CNV2*) when constructing B_k. Here, the critical points of the largest circle are chosen to form B_k only which are then allocated to the other clusters. $K_{\max}$ is therefore set to 2 or 3 as a circle is determined by two points ($K_{\max}$ = 2) or three points ($K_{\max}$ = 3) only. The only step which is changed here is *Step* 0 of Figure 3.3 which is replaced by:

Step 0: Specify $K_{\max}$, $\text{CPU}_{\max}$ and set $Time = 0$. Define the sequence of neighbourhood structures NC_k using $B_k \subset \overline{E}_c$ with $|B_k| = k$ instead of (3.2) with $\overline{E}_C$ being the set of the critical points defining the largest circle at a given iteration. Note that (3.1) and (3.3 – 3.5) remain unchanged

3.2.6 Computational results for the proposed customer-based VNS

In our implementation, we used a CPU time corresponding to 10,000 iterations of MSALA (Multi- Start Alternate Locate-Allocate Algorithm) as the stopping condition. This section presents computational results of the MSALA algorithm, the original VNS algorithm of customer-based neighbourhood VNS(*CN*) and its three enhancements (*CNV1*, *CNV2* and *CNV3*). The results were carried out for the unweighted case. Our chosen enhanced implementation of the Elzinga-Hearn algorithm described in chapter 2 (i.e., *V3* or *V4*) is used in our local search. All our algorithms were programmed in C++ using Visual Studio 2008. A laptop computer with an Intel Core 2 Duo processor, 2.0 GHz CPU and 4G memory was used to conduct these numerical experiments. The optimal solutions for the small existing data set (n=439 TSP-Lib) with p=10 to 100 that were provided by Chen and Chen (2009) were

also used to assess the performance of our customer-based neighbourhood approaches. Note that this is the only data set for which optimal solutions exist and hence our heuristics will be tested on larger instances in later experiments. In this study, we ran the original algorithm VNS(*CN*) and the three enhancements 10 times, starting from a random initial solution, using a total time for each run as $\frac{\text{CPU of 10,000 runs}}{10}$. The computational results were presented using the deviations of the best and the average results.

Deviations of the best results VNS(CN)

Table 3.1 consists of the optimal solution and the deviations of the best results from the optimal solution for the 10 runs. The CPU time (in seconds) for the algorithms was also given. We showed the differences in the solution quality between MSALA, the original algorithm of VNS(*CN*) and the other enhancements. The deviations (%) of these algorithms were computed from the optimal solutions as

$$\text{Deviation (\%)} = \frac{(Z_H - Z^*)}{Z^*}.100$$

where Z_H and Z^* refer to the objective function values of a given heuristic *'H'* and the optimal solutions respectively, where *'H'* refers to MSALA, *CNV1*, *CNV2* and *CNV2*.

Table 3.1: Deviation (%) from the optimal solution for MSALA, the original algorithm of VNS(*CN*), and its enhancements (*CNV1*, *CNV2* and *CNV3*)

n = 439 TSP-Lib	The optimal solutions (*Z*)	MSALA		The Original Algorithm VNS(CN)		Variant (*CNV1*)		Variant (*CNV2*)		Variant (*CNV3*)		Total Time (secs)
p		Deviation %	CPU (Time)*	Deviation %	CPU (Time)*	Deviation %	CPU (Time)*	Deviation %	CPU (Time)*	Deviation %	CPU (Time)*	
10	1716.5099	2.02	115.61	0	5.42	0	6.76	0	7.45	0	6.14	43.51
20	1029.7148	11.42	643.80	0	24.02	0	22.54	0	23.81	0	30.47	75.13
30	739.19297	31.90	22.22	0	37.50	0	24.90	0	20.04	0	30.01	101.90
40	580.00539	34.47	958.10	0	46.59	0	47.56	0	47.29	0	33.50	117.11
50	468.54162	39.88	456.16	2.98	70.63	2.67	90.99	2.67	78.61	2.67	81.23	173.01
60	400.19527	44.93	1216.84	6.11	89.23	2.64	115.90	2.92	53.85	2.46	62.43	198.42
70	357.94553	59.27	75.19	3.80	107.20	3.80	155.30	1.27	88.84	1.27	73.68	208.74
80	312.5	61.05	50.53	7.99	156.40	6.37	114.30	6.51	120.40	6.51	85.03	194.31
90	280.90256	73.78	1360.94	6.15	127.20	6.15	135.90	5.77	107.80	2.93	110	198.81
100	256.68019	66.58	1351.69	7.14	82.28	7.14	115.10	6.54	111.60	7.14	72.76	189.64
Average	614.21882	42.53	625.11	3.42	74.65	2.88	82.93	2.57	65.97	**2.30**	**58.53**	150.06

**CPU time when the best solution was found* (*recorded for information only*)

From Table 3.1, it can be noted that the deviation values increased with *p* for all the algorithms. We also conclude that the performance of the MSALA algorithm was relatively

poor compared to the original algorithm VNS(*CN*) and the three enhancements showing an overall average deviation of 42.53%, 3.42%, 2.88%, 2.57% and 2.23% respectively. Furthermore, it has been noted that the original algorithm and the three enhancements were able to find the optimal solution when $p \leq 40$, while MSALA failed to find even a single one. For comparison purposes between the original algorithm and the three variants, we can say that the latter outperformed the former. In general, we can confirm that *CNV3* was better than *CNV1* and *CNV2* in terms of solution quality and computational effort. The average CPU time when the best solution was found was also recorded for the original algorithm, *CNV1*, *CNV2* and *CNV3* as 74.65, 82.93, 65.97 and 58.53 secs respectively. This was shown just to illustrate that the best solution can be found much earlier and hence other stopping criteria could be derived instead. For instance the search could stop if no improvement is found after a few consecutive runs.

Deviations of the average result (*CN*)

To provide a more robust statistical analysis, we present the average results of the original algorithm and the variants *CNV1*, *CNV2* and *CNV3*.

Table 3.2 consists of the optimal solution and the deviations (%) of the average result from the optimal solution (10 runs) for the original algorithm VNS(*CN*) and the three enhancements. Furthermore, their respective standard deviations (ST DEV) were also shown.

Table 3.2 Deviation (%) of the average result from the optimal solution for the original algorithm of VNS(*CN*), and its enhancements (*CNV1*, *CNV2* and *CNV3*)

n = 439 TSP-Lib	The optimal solutions (*Z*)	The Original Algorithm VNS(CN)			Variant (*CNV1*)			Variant (*CNV2*)			Variant (*CNV3*)		
p		Average *Z*	Deviation %	ST DEV	Average *Z*	Deviation %	ST DEV	Average *Z*	Deviation %	ST DEV	Average *Z*	Deviation %	ST DEV
10	1716.5099	1723.980	0.44	12.03	1726.470	0.58	12.86	1731.450	0.87	12.86	1730.273	0.80	15.23
20	1029.7148	1041.512	1.15	24.88	1035.611	0.57	18.66	1039.267	0.93	30.22	1045.168	1.50	33.71
30	739.19297	742.069	0.39	7.76	741.308	0.29	6.69	742.912	0.50	11.42	744.795	0.76	11.94
40	580.00539	599.705	3.40	10.15	601.763	3.75	16.60	593.232	2.28	14.49	591.258	1.94	12.48
50	468.54162	502.483	7.24	24.88	492.574	5.13	9.12	488.237	4.20	9.22	489.390	4.45	9.19
60	400.19527	428.763	7.14	7.77	424.510	6.08	5	425.304	6.27	4.85	426.167	6.49	6.65
70	357.94553	375.252	4.84	3.40	376.792	5.27	4.49	377.436	5.45	6.82	374.216	4.55	7.68
80	312.5	348.744	11.60	8.70	343.835	10.03	8.71	343.282	9.85	11.28	344.880	10.36	10.80
90	280.90256	306.510	9.12	7.70	303.374	8	4.57	306.229	9.02	10.42	300.381	6.93	8.26
100	256.68019	282.615	10.10	5.54	280.834	9.41	5.56	279.768	9	4.84	282.165	9.93	6.62
Average	614.21882	635.163	5.54	11.28	632.707	4.91	**9.23**	632.712	4.84	11.64	632.869	**4.77**	12.26

The average results also confirmed that the performance of *CNV3* algorithm outperformed the other ones, where the average deviations from the optimal solution for the original algorithm, *CNV1*, *CNV2* and *CNV3* algorithms were 5.54%, 4.91%, 4.84% and 4.77% respectively. In general we can say that the standard deviation values decrease with *p* showing that the objective function values of *CNV1* were less spread than the ones found by *CNV3* (i.e., 9.23 and 12.26), although *CNV3* gave better results than the other ones.

3.3 The facility-based VNS(*FN*)

In this section, the facility-based neighbourhood algorithm, VNS(*FN*) for short, is presented followed by five enhancements on the original algorithm. Their steps are similar to those of the VNS(*CN*) given in Figure 3.3 except that in the shaking step, *k* open facility locations are selected randomly and inserted into other places. These facilities can be located either in the discrete space (demand points) or in the continuous space, as shown in Figure 3.8 (*a*). This figure shows a feasible solution of four clusters with facilities located at p_1, p_2, p_3 and p_4. Here, facility p_3 is located randomly in the continuous space as shown by $\bar{p}_3$. The point a_1 and d_1 are now closer to $\bar{p}_3$ than their original facilities located at p_1 and p_4 respectively.

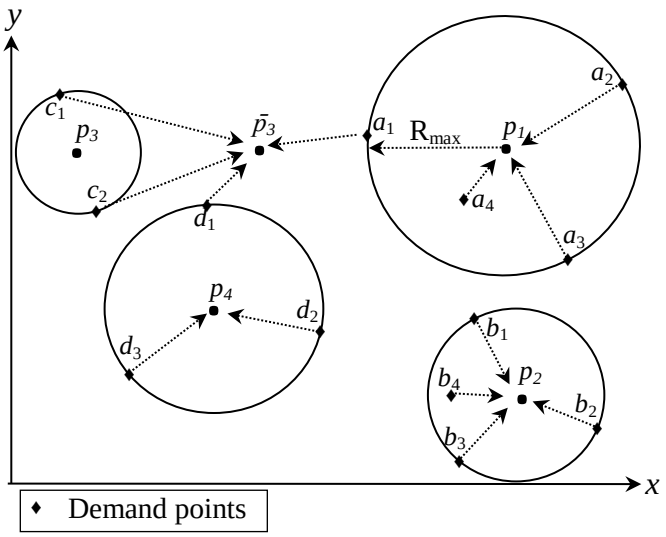


Figure 3.8 (*a*): A feasible solution of a *4*-centre location problem

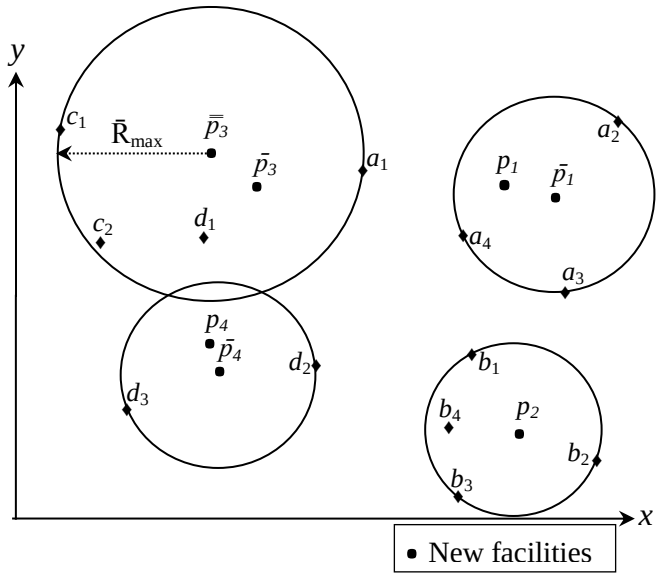


Figure 3.8 (*b*): A worse solution of the same problem

When the local search is applied, the new centre of the cluster at p_1 (a_2, a_3 and a_4) is moved to $\bar{p}_1$, the cluster (c_1, c_2, a_1 and d_1) with its initial centre at $\bar{p}_3$ is now moved to $\bar{\bar{p}}_3$ and p_4 (d_2 and d_3) is moved to $\bar{p}_4$, see Figure 3.8 (*b*). However, in this particular example a worse solution is generated as $\bar{R}_{max} > R_{max}$.

3.3.1 The original facility-based neighbourhood algorithm VNS(*FN*)

As the chosen *k* open facilities are inserted randomly either at demand points or in the continuous space, we therefore classify this type of neighbourhood, which we denote by $FN_k(X); k = 1,..., K_{\max}$, under two categories. We refer to these as VNS1(*FN*) and VNS2(*FN*). These are defined as follows:

Algorithm VNS1(*FN*):- *Remove k open facilities randomly from the current solution and replace them randomly at the fixed points*.

Here we define the k^{th} neighbourhood structure $FN_k(X)$; k=1,...,$K_{\max}$ as

$$FN_k(X) = \left\{ X' = X \setminus C_k^- \cup C_k^+ \text{ with } C_k^- = \bigcup_{r=1}^{k} X_r \text{ and } C_k^+ = \bigcup_{r=1}^{k} X_r' \text{ where } X_r \in X \text{ and } X_r' \in \{P_1,..., P_n\} \setminus X \right\} \quad (3.6)$$

The main steps of VNS1(*FN*) are similar to VNS(*CN*) of Figure 3.3 except that *Step* 0 and *Step 2a* are replaced as follows:

Step 0: Define $FN_k(X)$; k=1,...,$K_{\max}$ using (3.6) with $K_{\max} = \lceil \sqrt{p} \rceil$.

Step 2a : Generate $X' \in FN_k(X)$ using (3.6)

Algorithm VNS2(*FN*):- *Remove k open facilities randomly and insert them randomly in the continuous space.*

Here the k^{th} neighbourhood structure $FN_k(X); k = 1,..., K_{\max}$ is defined as follows:

$$FN_k(X) = \left\{ X' = X \setminus C_k^- \cup C_k^+ \text{ with } C_k^- = \bigcup_{r=1}^{k} X_r \text{ and } C_k^+ = \bigcup_{r=1}^{k} X_r' \text{ where } X_r \in X \text{ and } X_r' \in S \subset \Re^2 \right\}$$

$$\text{with } S = \{(x, y) \in \Re^2 : \underset{i=1,...,n}{Min}(a_i) \le x \le \underset{i=1,...,n}{Max}(a_i) \,\&\, \underset{i=1,...,n}{Min}(b_i) \le y \le \underset{i=1,...,n}{Max}(b_i)\} \quad (3.7)$$

VNS2(*FN*) is similar to VNS1(*FN*) except that *Step* 0 and *Step* 2*a* are replaced by

Step 0: Define $FN_k(X)$; k=1,...,$K_{\max}$ using (3.7) with $K_{\max} = \lceil \sqrt{p} \rceil$.

Step 2a: Generate $X' \in FN_k(X)$ using (3.7).

Computational result of VNS1(*FN*) versus VNS2(*FN*)

The existing data (n=439 TSP-Lib) with p =10 to 100 with an increment of 10 was also used here to assess the performance of these two variants. For each value of p we ran the approaches 10 times, starting from a random initial solution. Table 3.3 showed that the performance of VNS2(*FN*) was found to be better and relatively quicker than VNS1(*FN*). However, the objective function values of VNS1(*FN*) were much less spread than the ones found by VNS2(*FN*), where the ST Deviation values were 28.42 and 41.94 respectively. In general, we can conclude that VNS2(*FN*) outperformed VNS1(*FN*) in terms of the average and the best result, where the average deviations of the best result were 23.63% and 28.95% respectively. We therefore concentrate on proposing four simple but effective enhancements on VNS2(*FN*) which are described in the next subsection.

Table 3.3: Deviation (%) of the average and the best result from the optimal solution for both the original facility-based algorithms (VNS1 and VNS2)

n = 439 TSP-Lib / p	The optimal solutions (Z)	The original algorithm VNS1(*FN*) (Discrete case)				The original algorithm VNS2(*FN*) (Continuous case)			
		Deviation Average Results	Deviation Best Results	ST DEV	CPU Time*	Deviation Average Results	Deviation Best Results	ST DEV	CPU Time*
10	1716.510	2.85	1.45	24.64	16.92	1.43	0.61	8.85	23.34
20	1029.710	15.42	11.42	33.67	35.52	12.79	5.73	58.46	23.86
30	739.193	35.11	31.90	45.18	52.80	32.77	21.77	66.50	28.27
40	580.005	39.28	13	72.36	51.15	27.13	9.06	73.41	47.63
50	468.542	37.58	31.11	19.21	86.22	45.77	20.44	71.57	39.56
60	400.195	44.58	37.75	19.02	108.40	38.72	18.86	39.33	68.97
70	357.946	49.28	40.60	23.41	80.54	50.23	40.60	33.45	79.53
80	312.500	50.87	40.06	16.02	132.56	48.88	40	25.34	85.81
90	280.903	49.29	39.02	18.45	130.26	49.13	37.95	20.59	85.67
100	256.680	51.16	43.23	12.21	118.43	53.65	41.23	21.94	91.48
Average	614.219	37.54	28.95	**28.42**	81.28	**36.05**	**23.63**	41.94	**57.41**

*: *CPU time when the best solution is found.*

3.3.2 Some VNS2(*FN*) based enhancements

There are some steps in VNS2(*FN*), especially in the shaking phase of *Step 2a*, which are worth examining. We aim to shake with a strong perturbation, also known as 'Intensified shaking' in the literature, see Mladenovic *et al.* (2013). Therefore, in the next subsections, we will present several modifications in *Step* 2*a*, leading to four effective enhancements. For simplicity and for the purpose of clarification, this step (*Step* 2*a*) is divided into two phases as follows:

Facility Removal:- *the removal of the k open facilities*
In this phase, the *k* facility candidates for removal are chosen based on certain rule that will be explained next.

Facility Attraction:- *relocating the chosen facilities*
Here, the *k* open facilities that have been chosen for removal are located into other well defined destination regions that we will describe later.

A) VNS2(FNV1): The allocation of facilities between the small circles and the larger ones

Since the objective function of the *p*-centre aims to minimise the maximum distance (R_{max}), the first idea which comes to one's mind is to reallocate the facilities with small circles and insert them randomly in the larger ones. Therefore, in this variant, we first order the facilities (circles) in descending order of their radii. We then choose *k* open facilities randomly from the bottom half (i.e., the smaller circles) and locate them also randomly in the continuous space of the larger circles (i.e., circles of the top half), where each of the *k* added facilities is located in a separate circle.

The main steps of this enhancement are similar to the original algorithm (VNS2(*FN*)) except that *Step* 2*a* is replaced by:

Step 2a:

(i) *Sorting the Facilities*:
Sort the facilities (circles) in descending order of their radii. Make up two groups as follows: (G_1 as the set of the larger circles) with $|G_1| = [p/2]$ and G_2 the set of the smaller circles with $|G_2| = p - |G_1|$.

(ii) *Facility Removal:*
Choose randomly *k* open facilities from G_2.

(iii) *Facility Attraction:*

Locate randomly the k facilities separately, each in the continuous space encompassed by the larger circles in G_1.

B) VNS2(FNV2): Largest circle-based removal and relocation

When the solution of the *p*-centre location problem is not optimal, it is observed that the facility in the largest circle and at least one of its neighbouring facilities cannot be in the right location. This observation led us to explore the idea of reallocating the facility of the largest circle and the facility locations that are around it. The region that we choose the k open facilities from is called the covering circle, which we refer to as $CC_{k'}$. This is a circle with a dynamically increasing radius, from the centre of the largest circle to its $(k')^{th}$ nearest facility. For the sake of simplicity let's index the largest circle as C_1. This is defined by (X_1, R_1) with X_1 as its centre and R_1 as its radius. The remaining $p-1$ circles are indexed in ascending order of their distances from the largest circle using the distance measure $d(X_j, X_1); j = 2,..., p$. The following notation is used.

Notation

C_j: the j^{th} nearest circle to the largest circle $C_1; j = 2,..., p$

$\hat{C}_j$: the area encompassed by circle $C_j; j = 1,..., p$

$CC_{k'}$: the k' facilities encompassed by the artificial circle centred at X_1 with a radius

$\hat{R}_{k'} = d(X_{k'}, X_1)$ if $k' > 1$ and $\hat{R}_1 = R_1$ otherwise (i.e., $k' = 1$); $k' = 1,..., p$.

We refer to $CC_{k'}$ as the k'^{th} covering circle. This can also be defined as a sequence $\{CC_{k'}\} = \{X_1,..., X_{k'}\}$ representing the facility of the largest circle and the $k'-1$ nearest facilities to it.

The steps of this enhancement which we call VNS2(*FNV2*) are given in Figure 3.9, which also contains the updating of the $CC_{k'}$. In general, these steps are similar to the original algorithm VNS2(*FN*) except that *Step* 2*a* is replaced as follows:

Facility Removal:

In the k^{th} neighbourhood, instead to choose k facilities randomly from $X = (X_1,..., X_p)$ we choose these facilities from $CC_{k'}$ where $X_j \in CC_{k'}$; j=1,…,k and $k \le k'$ where k' is the level at that iteration, $k' = 1,..., p$, see Figure 3.9. The way the value of k' is updated is defined next.

Facility Attraction:

These k removed facilities are located randomly in the continuous space encompassed by the k larger circles separately. For instance, when k=2 we locate the first facility in the area encompassed by the 1st largest circle (C_1) also known and the second one in the region of the 2nd largest circle randomly.

The updating of $CC_{k'}$

As the removal process of the k facilities and their insertion are linked to VNS and to the corresponding covering circle $CC_{k'}$ at a given iteration, we briefly describe how the value of k' is updated. This is also given in the algorithm that follows in Figure 3.9. We first remove a facility from CC_1 namely the facility encompassed by the largest circle, this facility is then located randomly in CC_1. The local search is then applied on this perturbed solution. If the solution is not improved, we remove 2 facilities from CC_2 and insert them randomly in CC_2. This process is repeated until we reach $CC_{K_{\max}}$. At this iteration if there is no improvement we revert back to $k = 1$ as in the standard VNS but we continue increasing k' by setting $k' = K_{\max} + 1$ instead. We also continue increasing the radius of the covering circle until we either reach CC_p (note that k can be any value between 1 and $K_{\max}$ but $k' = p$) or an improved solution is found where we revert back to $k = k' = 1$. If the latter case happens, we decrease the radius of $CC_{k'}$ by setting $k' = k' - 1$ where we remove $k = k + 1$ facilities from $CC_{k'} = CC_{p-1}$ and so on until we reach CC_1. However as $k \le k'$, to control the increase and the decrease of k' we introduced an indicator which we call $Flag$. If $Flag = 1$ the covering circle is increasing $(k' = k' + 1)$, otherwise it is decreasing $(k' = k' - 1)$. However if at any iteration we have $k \ge k'$, we then reset $k = k'$ and $Flag = 1$. As we start with CC_1 we always initialise $Flag$ to 1.

Based on the neighbourhood structure described earlier and the way $CC_{k'}$ is updated, the VNS2(*FNV2*) algorithm is summarised in Figure 3.9.

Step 0: Specify $K_{\max}$, $\text{CPU}_{\max}$ and set Time=0. Define the neighbourhood structures $FN_k; k = 1,..., K_{\max}$

Step 1: Generate an initial feasible solution X, record the objective function $Z(X)$ and set $k = k' = 1$ and $Flag = 1$

Step 2:

- *Step* 2*a*: (i) If $k > k'$ set $k' = k$ and Flag=+1
  (ii) Generate $X' \in FN_k(X)$ ***"Shaking Step"***
- *Step* 2*b*: Apply a local search to obtain X'' starting from X' ***"Local Search Step"***
- *Step* 2*c*: ***"Evaluation Step"***

  If $Z(X'') < Z(X)$ set $X = X''$, $Z(X) = Z(X'')$, $k = 1$, $k' = 1$ and $Flag = 1$

  Otherwise

  If $k = K_{\max}$ set $k = 1$, else set $k = k+1$

  If $Flag = 1$ **"*The update of* $CC_{k'}$"**

  if $k' < p$ then set $k' = k' + 1$, else $Flag = -1$

  Else

  if $k' > 1$ then set $k' = k' - 1$, else $Flag = +1$

Step 3: Stopping condition.

If $Time > CPU_{\max}$ we record the incumbent solution (X) and stop, else go to *Step* 2.

Figure 3.9: The VNS2(*FNV2*) algorithm

As an example in Figure 3.10, from CC_1 we select facility p_1 to locate randomly in the area encompassed by CC_1. If the local search improves the solution, we will record the new solution and start again from the new CC_1; otherwise we explore CC_2 where we have two facilities p_1 and p_2. These will be located randomly in the largest and the second largest circles centred at p_1 and p_5 separately.

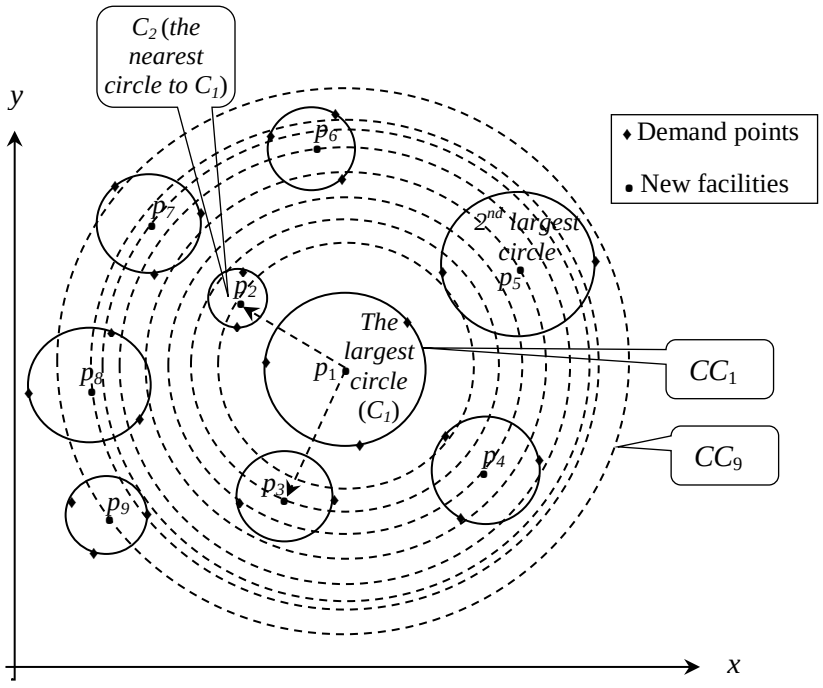


Figure 3.10: An example of the levels of covering circles that are dynamically increasing from the source region (i.e., CC_1,...,CC_9)

C) VNS2(FNV3): Controlling the facility insertion in VNS2(FNV2) using covering circles

The idea of this enhancement is to modify VNS2(*FNV2*) in the location of the *k* removal facilities. Here, we locate randomly the *k* removed facilities in the area encompassed by the $CC_{k'}$. The facility removal phase of *Step* 2*a* is unchanged as in VNS(*FNV2*) but the facilities attraction is replaced as follows:

Facility Attraction:

In the k^{th} neighbourhood instead to locate $k = 1,\ldots,K_{\max}$ facilities randomly in the larger circles we insert these facilities randomly in $CC_{k'}$ with k' being the level at that iteration, $k' = 1,..., p$.

For instance, in Figure 3.10, from the first level of the covering circle (CC_1), facility p_1 is selected to be located randomly again in the area encompassed by CC_1. If the local search improves the solution, the new solution is recorded and the search starts again from the CC_1; otherwise the second level of the covering circle namely CC_2 is explored where two facilities located at p_1 and p_2 are removed. These two facilities are then located randomly again in the continuous space encompassed by the same covering circle namely CC_2.

D) VNS2(FNV4): Location based on the critical points regions

In the *p*-centre problem, there is a number of regions around each open facility that cannot contain any facility that could be worth opening. The following additional notation is used.

Additional notations

$CP_j = \{P_i \in W_j : d(P_i, X_j) = R_j; i = 1,..., n\}$: the set of critical points of

C_j ($|CP_j| \leq 3$); $j = 1,..., p$ with W_j defined in (3.1).

RC_{jl}: the area encompassed by the circle centered at $l \in CP_j$ with radius R_j; $j = 1,..., p$

$CR_j = \hat{C}_j \cup \bigcup_{l \in CP_j} RC_{jl}$: the j^{th} critical region made up of $\hat{C}_j$ and its $|CP_j|$ surrounding

RC_{jl} ($l \in CP_j$); $j = 1,..., p$

$UCR_k = \bigcup_{j=1}^{k} CR_j; k = 1,..., K_{\max}$: the union of the k critical region

For example, Figure 3.11 (*a*) shows two critical points regions (i.e., RC_{1l} with l representing the critical points a_1 and a_2). It can be shown that these two regions could not contain any facility worth considering. This is because if one of these regions contained a facility, the point of that region would have been already allocated to this facility. For instance, if the region of point a_1 contained a facility, a_1 would be closer to this facility than its serving facility located at p_1, and therefore the point a_1 would have already been allocated to that facility instead. This idea is similar to the interesting and powerful property given and proved in Mladenovic *et al*. (2003) and Drezner (1984*b*) for the discrete and the continuous cases respectively. Figure 3.11 (*b*) shows the same case for a circle defined by three critical points, which contains three regions (i.e., RC_{1l} with l representing the critical points a_1, a_2 and a_3) that also cannot contain any facility.

In the preliminary study, we observed that when a new facility is inserted randomly in the area encompassed by the largest circle, the size of the circle will decrease. This is because at least one of its critical point will be allocated to the new facility. In addition, locating the new facility in one of the regions defined by the critical point (i.e., the regions with point a_1 (RC_{1a_1}) and a_2 (RC_{1a_2}) in Figure 3.11 (*a*)), the radius of the circle centred at p_1 will decrease.

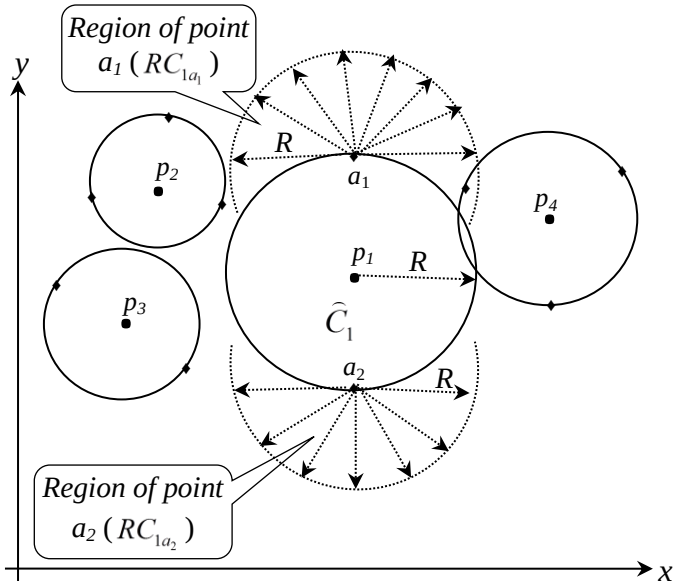


Figure 3.11 (*a*): An example of 2 regions that do not contain any facility for a circle defined by 2 critical points (a_1, a_2)

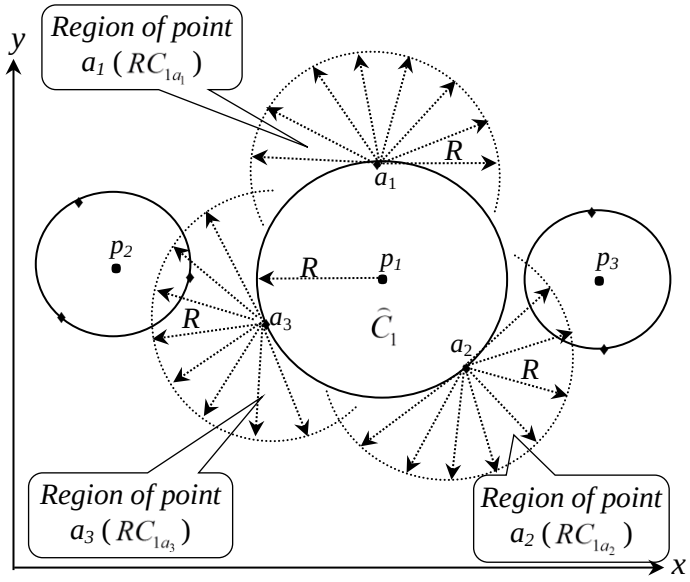


Figure 3.11 (*b*): An example of 3 regions that do not contain any facility for a circle defined by 3 critical points (a_1, a_2, a_3)

However, locating a new facility in one of the regions of these critical points will give more chance to reduce the radius of the other circles. This is because the new facility will be closer to the other facility than if we locate it inside the circle centred at p_1. This can attract some customers of the other clusters to be allocated to the new facility.

We take this observation into account to define our neighbourhood for attracting facilities to produce our final enhancement VNS2(*FNV4*). This is achieved by exploring those regions defined by RC_{jl} as the regions where a facility could be located; $j = 1,..., p$ and $l \in CP_j$. In general, the steps of VNS2(*FNV*4) are very similar to those of VNS2(*FNV*3) given in Figure 3.9, except that the second phase (*Facility Attraction*) of *Step* 2*a* is replaced by the following:

Facility Removal:

This step is the same as the step of VNS2(*FNV3*).

Facility Attraction:

In the k^{th} neighbourhood, we insert randomly these k chosen facilities in $UCR_{k'}$ where k' is the level at that iteration, $k' = 1,..., p$. Each facility is located randomly in the continuous space encompassed by CR_j; j=1,…,p.

In this enhancement, the new k^{th} neighbourhood structure that combines the facility attraction and the facility removal is defined as follows:

$$FN_k(X) = \{X \setminus \bigcup_{r=1}^{k} X_r \cup \bigcup_{r=1}^{k} X'_r \;;\; k = 1,...,K_{\max}\}$$

Where $(X_1,..., X_k) \in CC_{k'};\ k \le k';k' = 1,..., p$ and $(X_1^{'},..., X_k^{'}) \in UCR_k$ with the j^{th} facility being located in the continuous space delimited by $CR_j;\ j = 1,..., k$

For instance, Figure 3.12 shows the fourth level of the covering circle CC_4 (k'=4), which contains four facilities located at p_1, p_2, p_3 and p_4. Figure 3.12 also shows the areas (UCR_4) where we have to insert the k chosen facilities. This includes $\widehat{C}_j$ ($\widehat{C}_1, \widehat{C}_2, \widehat{C}_3, \widehat{C}_4$), $RC_{1l}(a_1, a_2, a_3)$, $RC_{2l}(b_1, b_2)$, $RC_{3l}(c_1, c_2, c_3)$ and $RC_{4l}(d_1, d_2)$. All these regions could be used as destination areas for inserting the k facilities (FN_k; k=1,…,$K_{\max}$).

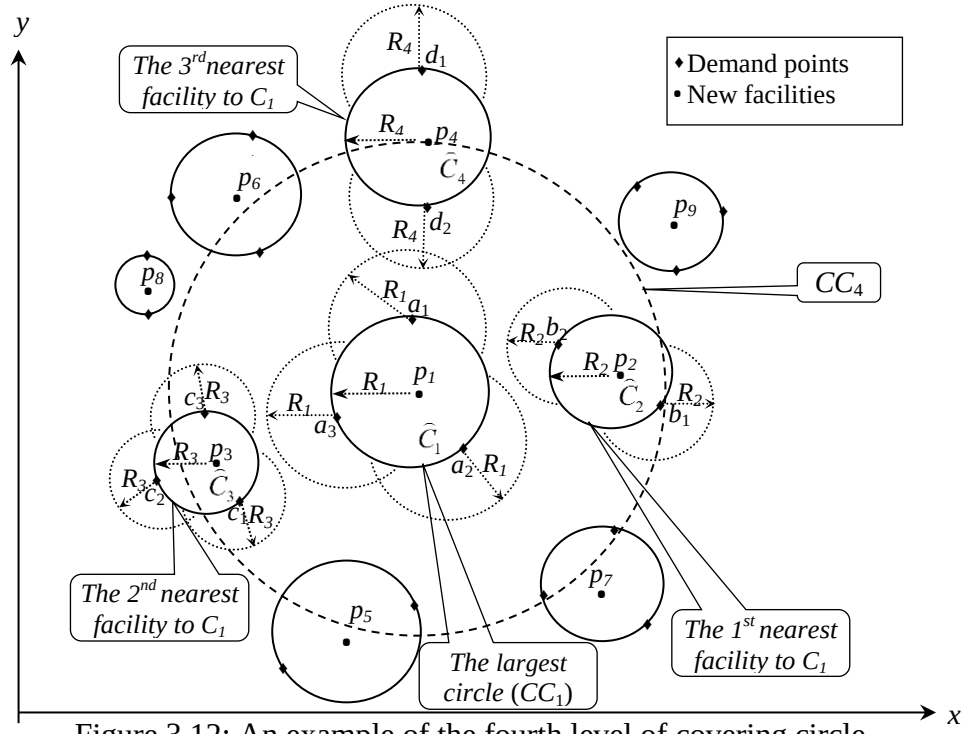


Figure 3.12: An example of the fourth level of covering circle with its critical points regions (the destination regions)

For example, if k=3, we have to choose randomly three facilities among the facilities that are in CC_4 (p_1, p_2, p_3 and p_4), these k chosen facilities are then relocated randomly again in the destination areas, namely each chosen facility is located randomly in CR_j separately (j=1,…,4)

3.3.3 Computational results using VNS based (*FN*)

In this section, we present computational results of the MSALA algorithm, the original VNS for facility-based (*FN*) and the four enhancements. The existing data (n=439 TSP-Lib) with p=10 to 100 was used to assess the performance of these variants. Here, we also ran the original algorithm (*FN*) and the four enhancements 10 times, starting from a random initial solution, using a total time for each run as $\frac{\text{CPU of 10,000 runs}}{10}$.

Deviations of the best results

The deviations (%) from the optimal solutions and the CPU time (in seconds) when the best solution was found were shown in Table 3.4.

Table 3.4: Deviation (%) of the best result from the optimal solution for MSALA, the original algorithm of VNS2(*FN*), and its enhancements (VNS2(*FNV1*),..., VNS2(*FNV4*))

n = 439 TSP-Lib	The optimal solutions (Z)	MSALA		The Original Algorithm VNS2(*FN*)		Variant VNS2(*FNV1*)		Variant VNS2(*FNV2*)		Variant VNS2(*FNV3*)		Variant VNS2(*FNV4*)	
p		Deviation %	CPU Time*	Deviation %	CPU Time*	Deviation %	CPU Time*	Deviation %	CPU Time*	Deviation %	CPU Time*	Deviation %	CPU Time*
10	1716.5099	2.02	115.61	0.61	23.34	1.37	15.20	0	11.11	0	4.53	0	9.20
20	1029.7148	11.42	643.80	5.73	23.86	0	27.34	0	8.10	0	31.38	0	17.10
30	739.19297	31.90	22.22	21.77	28.27	0.81	35.89	0	18.23	0	12.82	0	28.76
40	580.00539	34.47	958.10	9.06	47.63	4.54	55.87	0	70.87	0	44.60	0	39.77
50	468.54162	39.88	456.16	20.44	39.56	14.19	94.35	2.67	103.34	1.03	92.37	2.11	92.06
60	400.19527	44.93	1216.84	18.86	68.97	14.98	97.32	3.62	90.67	3.62	65.01	3.27	106.20
70	357.94553	59.27	75.19	40.60	79.53	19.45	75.47	2.71	158.54	3.43	114.71	1.27	124.39
80	312.5	61.05	50.53	40	85.81	22.70	130.39	6.47	163.39	6.51	152.47	6.51	240.67
90	280.90256	73.78	1360.94	37.95	85.67	28.12	122.47	3.50	133.26	3.85	102.65	3.79	166.33
100	256.68019	66.58	1351.69	41.23	91.48	28.57	134.58	7.14	122.40	4.90	127.70	6.54	121.70
Average	614.21882	42.53	625.11	23.63	57.41	13.47	78.89	2.61	87.99	2.33	74.82	2.35	94.62

*: *CPU time when the best solution was found*

In general, the computational results showed that the deviation values increased with p for all versions. We can conclude that the original approach (VNS2(*FN*)) outperformed the multi-start approach, where their overall average deviation values of the best result were 23.63% and 42.53% respectively. For the enhancements, all their performances were much better than the original algorithm VNS2(*FN*). However, it can be noted that there was a significant improvement in the solution in the last three variants VNS2(*FNV2*), VNS2(*FNV3*) and VNS2(*FNV4*), where their overall average deviation values were 2.61%, 2.33%, and 2.35% respectively. In addition, these three variants found the optimal solution when $p \leq 40$, while

the second variant found the optimal solution 3 times when $p \leq 30$, the first variant found the optimal solution only once, whereas the original algorithm VNS2(*FN*) cannot find any. In general, we can confirm that VNS2(*FNV3*) and VNS2(*FNV4*) were the best performers compared to the others in terms of solution quality with VNS2(*FNV3*) was slightly quicker than VNS2(*FNV4*).

Deviations of the average results

This section gives more details in terms of statistical analysis (deviations of the average results and ST deviations), in order to determine which of these two variants (VNS2(*FNV3*) and VNS2(*FNV4*)) perform better. Table 3.5 shows the optimal solution, the deviations (%) of the average result from the optimal solution for each algorithm (10 runs) and the standard deviation (ST Deviation).

Table 3.5: Deviation (%) of the average result from the optimal solution for the original algorithm (VNS2(*FN*)) and its enhancements (VNS2(*FNV1*),..., VNS2(*FNV4*))

n = 439 TSP-Lib	The optimal solutions (*Z*)	The original algorithm VNS2(*FN*)		Variant VNS2(*FNV1*)		Variant VNS2(*FNV2*)		Variant VNS2(*FNV3*)		Variant VNS2(*FNV4*)	
p		Deviation %	ST DEV	Deviation %	ST DEV	Deviation %	ST DEV	Deviation %	ST DEV	Deviation %	ST DEV
10	1716.5099	1.43	8.85	1.83	5.39	0.58	12.86	0.44	12.03	0.88	15.92
20	1029.7148	12.79	58.46	7.45	58.07	0	0	0.57	18.66	0	0
30	739.19297	32.77	66.50	3.66	21.80	0	0	0.46	10.72	0.02	0.49
40	580.00539	27.13	73.41	9.39	15.96	1.06	9.48	1.82	13.60	1.41	12.58
50	468.54162	45.77	71.57	19.62	13.18	3.23	5.98	4.65	11.67	3.98	8.57
60	400.19527	38.72	39.33	20.34	19	6.28	3.87	6.78	6.55	5.68	5.19
70	357.94553	50.23	33.45	22.46	10.44	4.79	5.50	6.22	8.67	3.36	6.93
80	312.5	48.88	25.33	31.82	18.27	8.56	5.96	9.09	7.96	10.01	10.64
90	280.90256	49.13	20.59	33.57	7.49	6.40	5.70	8.83	6.79	5.76	3.27
100	256.68019	53.65	21.94	32.84	13.53	9.44	6.05	10.94	8.97	8.96	6.27
Average	614.21882	36.05	41.94	18.30	18.31	4.03	5.54	4.98	10.56	**4.01**	6.99

In terms of average results, the performance of VNS2(*FNV4*) was found to be better than the other approaches with an overall average deviation of just over 4%. It can be noted that the computations of the best variant of the customer-based VNS namely *CNV3* were much quicker and its solution quality slightly better than VNS2(*FNV4*). The overall average deviation was 2.30% vs 2.35% and the average CPU time when the best solution was found was 58.53 vs 94.62 seconds respectively. However, the objective function value of VNS2(*FNV4*) was much less spread than the ones found by *CNV3*, (6.99 vs 12.26).

3.4 Enhancements on the allocation phase (local search)

The second part of the Cooper's locate-allocate procedure (i.e., the allocation phase) is also modified here. We propose two enhancements to be used when there is no improvement after the exchange between the location and the allocation phases. These include the allocation of the critical points and the closure of the non-promising facilities.

3.4.1 Allocation of a critical point of the largest circle to another facility

Since the aim of the *p*-centre problem is to minimise the maximum distance (R_{max}) and the largest circle can be determined by three or two critical points, allocating one of these critical points to another cluster (destination cluster) can improve the solution. This is possible as long as the new radius of the destination cluster is less than R_{max}. Here we focus on a simple but effective reallocation of the critical points of the largest circle to their neighbouring facilities.

Additional notations

C'_l = set of facilities encompassed by the circle $C(l, 2R_{\max}), l \in CP_1$

C''_l = set of facilities encompassed by the circle $C(l, R_{\max}), l \in CP_1$

$V_l = \{X_j \in C'_l \setminus C''_l; j = 1,..., p\}, l \in CP_1$: the set of facilities that are encompassed by C'_l but not by C''_l

The reasoning behind this enhancement is to remove a critical point $(l \in CP_1)$ and reallocate it in the neighbouring facilities that surround point l based on the subset V_l. This is performed for all $l \in CP_1$. The main steps of this procedure, which we refer to ALLOC, are given in Figure 3.14.

Note that in case there is more than one largest circle (case of tie) the procedure is repeated. This allocation process continues until a better allocation cannot be found.

For instance, Figure 3.13 (*a*) shows a feasible solution of a *5*-centre location problem with facilities located at p_1, p_2, p_3, p_4 and p_5. Here, p_1 is the location of the facility of CC_1 (the largest circle) with radius R_{max}. This circle is determined by three critical points (a_1, a_2 and a_3) representing CP_1. Figure 3.13 (*a*) also shows C'_{a_1} and C''_{a_1} based on the critical point a_1,

initially served from facility located at p_1. There are three facilities that are located at p_2, p_3 and p_4 in the region of V_{a_1}. If we allocate a_1 to one of these three facilities we can improve the solution as long as the new radius of the destination circle is less than the previous radius of the largest circle (R_{max}).

On the other hand, any facility outside C'_{a_1} (the second level of the critical point) cannot be used to improve the solution even if its radius = 0 (i.e., a circle containing one customer only). This is because the new radius of the destination cluster will be greater than the previous radius of the largest circle (R_{max}). For instance in Figure 3.13 (*a*), if we allocate the critical point a_1 to facility located at p_5 (contains one facility), the new radius of facility located at p_5 will be greater than R_{max}, because half the distance between p_5 and a_1 will be greater than R_{max}. Furthermore, the cluster that has its facility situated between C''_{a_1} and C'_{a_1} and which has one or more customers outside C'_{a_1} also cannot be used to improve the solution. For instance, facility at p_3 has customer c_1 outside C'_{a_1}, as shown in Figure 3.13 (*a*). This facility cannot improve the solution, because the distance between customer c_1 and the midpoint of $\overline{a_1 c_1}$ will be greater than R_{max}.

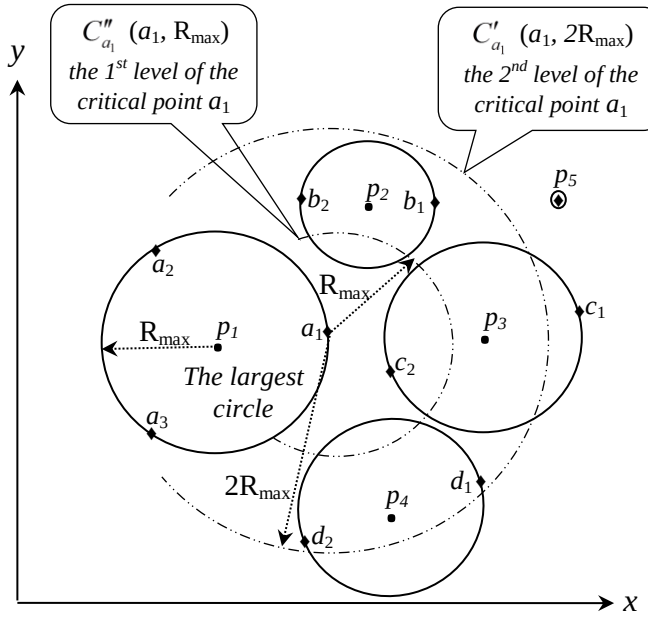


Figure 3.13 (*a*): The first and the second levels of allocating the critical points of the largest circle

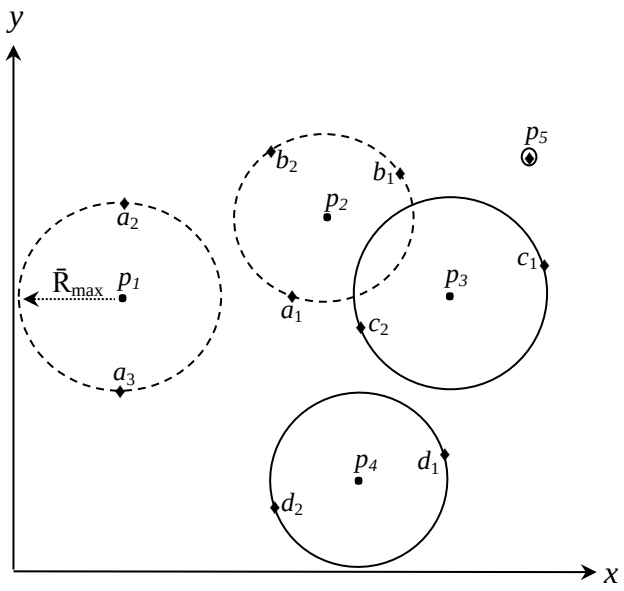


Figure 3.13 (*b*): A better solution of the same problem by allocating a critical point of the largest circle

However, not all the facilities that are within C'_{a_1} (even if all their customers belong to C'_{a_1}) can be used to improve the solution. This is because the new radius can be greater than R_{max} as in the example of facility at p_4.

It is worth noting that some of the facilities that exist in the region of V_{a_1} (p_2, p_3 and p_4) can be used to improve the solution as the radius of the new circle can be less than R_{max}. Figure 3.13 (*b*) shows the case where the critical point a_1 is allocated to facility at p_2 and the new radius ($\bar{R}_{max}$) becomes less than R_{max}.
The steps of the procedure (ALLOC) are summarised in Figure 3.14.

Step 1: Set $R' = R_{max} = R_1$

For each $l \in CP_1$ // set of critical points of the largest circle

{

Step 2: Solve the 1-centre problem for the largest circle without *l* and record its new radius $\bar{R}(l)$.

Step 3: Construct $C_l^{'}$, $C_l^{"}$ and V_l

Step 4: For each $j \in V_l$ (with $|V_l| > 0$) do the following: {

(i) Allocate *l* to facility *j*

(ii) Solve the *1*-centre problem for region *j* and record its radius R_j

(iii) If $R_j < R_{max}$

if $\bar{R}(l) > R_j \,\&\, \bar{R}(l) < R'$ set $R' = \bar{R}(l), l' = l$ and $j' = j$

else $R_j < R'$ then $R' = R_j, l' = l$ and $j' = j$ }

}

Step 5:

if $R_{max} > R'$ set $R_{max} = R'$ and record l' (chosen critical point) and j' (the facility attracting l')

Figure 3.14: The allocation procedure (ALLOC)

To illustrate the impact of this reallocation, computational results of the Multi-Start algorithm using 1000 runs with and without this enhancement were reported in Table 3.6. The existing data set with known optimal solutions ($n = 439$ TSP-Lib) with $p = 10$ to 100 was used here.

Table 3.6: Effect of the enhancement (based on 1000 runs of Multi-Start)

n= 439 TSP-Lib	Multi-Start (MSALA)		Multi-Start + ALLOC procedure			
			Objective function & CPU		Improvement Deviation (%)	
p	Z	CPU Time	Z	CPU Time	Z	CPU Time
10	1803.12	55.26	1753.08	55.28	2.78	0.03
20	1140.29	77.22	1125.28	77.26	1.32	0.05
30	975	91.13	975	91.13	0	0
40	822.34	123.08	760.35	123.12	7.54	0.03
50	739.19	133.44	698.77	133.55	5.47	0.08
60	635.04	146.09	570.09	146.27	10.23	0.12
70	570.09	160.30	570.09	160.40	0	0.06
80	570.09	168.24	542.71	168.30	4.80	0.04
90	570.09	175.74	570.09	175.74	0	0.00
100	503.27	196.79	437.68	196.99	13.03	0.10
Average	832.85	132.73	800.31	132.80	**4.52**	**0.05**

The integration of this reallocation procedure has improved the solution by up to 13% (when $p = 100$), with an average of over 4.5% while requiring a negligible extra computing time.

It is worth noting that the neighbouring facilities could also be identified using a Voronoi diagram, see Preparata and Shamos (1985). Though constructing the Voronoi diagram is polynomial, this could require a larger time as the construction of the Voronoi diagram needs to be performed at every iteration, as the facility locations change from one iteration to the next. For such reasons, we have opted for this simple but effective allocation method.

3.4.2 Removal of the non-promising facilities

The idea is to identify those facilities that serve the critical demand points only and to allocate these points to other facilities which will lead to such facilities having no customers and hence a reduction in the number of facilities. Let q be the number of facilities saved. These q facilities are then located one at time in the continuous space encompassed by the larger circles.

$$Let\ \delta_j = \begin{cases} 1 & \text{if } |W_j| = |CP_j|; \\ 0 & \text{otherwise,} \end{cases} \quad j = 1,..., p$$

For instance, Figure 3.15 (*a*) shows a feasible solution of a *5*-centre location problem. Here the critical points of the circle centred at p_3, namely c_1, c_2 and c_3 are allocated to the facilities located at p_5, p_4 and p_2 respectively. Note that there are no non-critical points encompassed by this circle. A feasible solution of a *4*-centre for the same problem is then shown in Figure 3.15 (*b*), where the new radius $R_1 = R_{max} = \bar{R}_{max}$. The facility initially located at p_3 can now be relocated in the largest circle centred at p_1 leading to having two facilities, each with a radius $\leq R_{max}$.

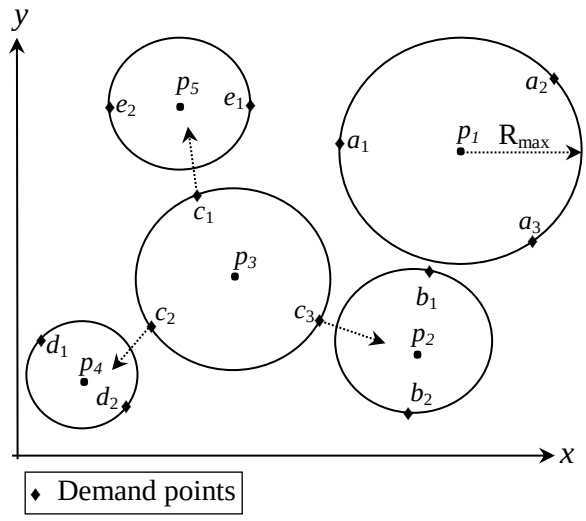


Figure 3.15 (*a*): A feasible solution of a *5*-centre location problem (*removal of the circle with centre p_3*)

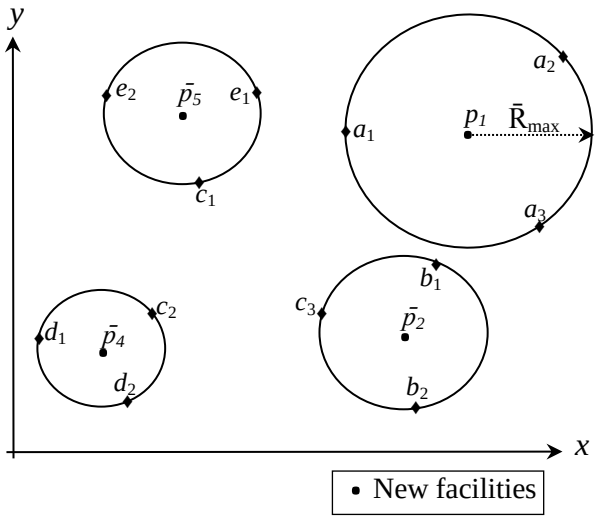


Figure 3.15 (*b*): The same objective function value but for its corresponding *4*-centre location problem (*Step* 2 of Figure 3.14)

Our preliminary study shows that this enhancement becomes more efficient when *p* is large as there is an opportunity to reduce the number of unnecessary facilities. Therefore, we propose two strategies for locating these saved facilities, which are as follows:

– *Allocating all the saved of facilities in one step*

Here, we are locating all the saved facilities, in the largest circles in one go. For example, if we have saved *q* facilities, these facilities will be located randomly in the continuous space of the *q* largest circles, where each of the *q* facilities is located in a separate circle. The local search is then applied.

– *Allocating the saved number of facilities one at a time*

Here, we are locating all the facilities one by one in the largest circle. Namely, one facility is located in the largest circle followed by the local search when the new largest circle is identified. The process is then repeated by locating a facility to the new largest circle. We continue this process until all the saved facilities are located.

The steps of the removal procedure are summarised in Figure 3.16, where the latter strategy of allocating the saved number of facilities one at a time is used in *Step* 3.

Step 1: set $q = 0$ and record C_1 //number of empty circles

Step 2: For each facility j with $\delta_j = 1; j = 1,..., p$ do

{

For $l \in CP_j$ do {

(i) Construct V_l

(ii) For each $r \in V_l$ assign *l* to facility *r* and solve the *1*-center problem of the affected region to yield R_r.

(iii) If $\underset{r \in V_l}{Min} R_r \geq R_{\max}$ go to the next *j* // delete facility *j*

Else set $s = Arg \underset{r \in V_l}{Min} R_r$ and update temporarily the radius and the centre of the affected circle *s* }

Facility *j* is closed and *q=q+1*

}

Step 3: If $q = 0$ there is no change and stop

Else

For $t = 1,..., q$

(i) locate the t^{th} closed facility randomly in $\hat{C}_1$

(ii) Apply the 'locate-allocate procedure' to find the new solution for the $p - q + t$ centre problem, and update C_1 if necessary.

Figure 3.16: The removal procedure of the non-promising circles

To illustrate the differences in the performance between these two approaches, the computational results of MSALA algorithm using 1000 runs (without applying the previous procedure (ALLOC)) was presented in Table 3.7. This table shows the number of facilities that has been saved, the deviation (%) in the objective function and the CPU time for both strategies.

Table 3.7: Results of the Multi-Start for 1000 iterations with and without the removal-based enhancement

n= 439 TSP-Lib	Multi-Start (MSALA)		Multi-Start + Removal Enhancement								# Saved Facilities
			One step				One by one				
			Objective function & CPU		Improvement Deviation (%)		Objective function & CPU		Improvement Deviation (%)		
p	*Z*	CPU Time	*Z*	CPU* Time	*Z*	CPU* Time	*Z*	CPU* Time	*Z*	CPU* Time	
10	1753.08	48.09	1753.08	48.10	0	0.03	1753.08	48.42	0	0.69	0
20	1226.02	71.79	1226.02	71.90	0	0.14	1226.02	71.84	0	0.07	0
30	975	92.00	975	92.10	0	0.11	975	92.03	0	0.03	0
40	975	107.66	975	107.76	0	0.10	975	109.06	0	1.30	0
50	834.74	141.20	813.37	141.34	2.56	0.10	822.34	141.42	1.49	0.16	1
60	655.39	167.27	631.50	167.78	3.65	0.31	631.50	167.54	3.65	0.16	1
70	580.01	175.50	551.28	175.87	4.95	0.21	503.27	176.02	13.23	0.29	3
80	570.09	178.62	453.47	179.22	20.46	0.34	459.62	179.25	19.38	0.35	5
90	570.09	190.11	475.16	190.41	16.65	0.16	459.96	190.94	19.32	0.44	6
100	503.27	192.67	400	192.98	20.52	0.17	332.84	194.03	33.87	0.71	7
Average	864.27	136.49	825.39	136.75	6.88	0.17	813.86	137.06	**9.09**	0.42	2.3

*: *CPU time when the best solution was found*

The first method was found to be slightly quicker than the second one where the overall average deviations for the CPU time were 0.17% and 0.42% respectively. But the latter was more efficient when these enhancements were introduced using the solutions of MSALA, where the average improvement in the solution were 6.88 % and 9.09% respectively. This is because in the first method the local search was applied only once, while in the second one the local search was used several times, amounting to the number of facilities that have been saved. This obviously provided more opportunities for the second method to improve the solution while requiring more computational time. For instance, it can be noted that the solution has improved by 33.87% when p=100. In general, it can be noted that the efficiency of this enhancement increased with p, as shown in Figure 3.17.

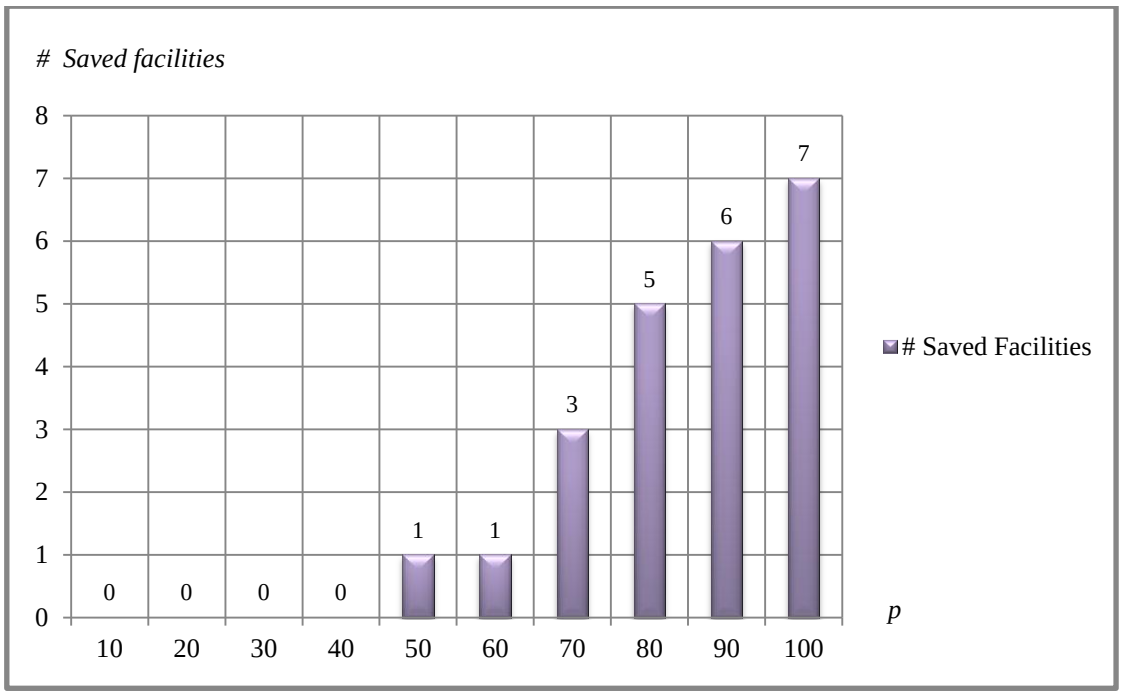


Figure 3.17: The number of saved facilities (from p = 10 to 100)

3.5 Incorporating learning within the *FN*-based VNS

In this section, we incorporate learning into our facility-based VNS heuristic. The aim is to identify any useful values of the parameters that are worth controlling in the search such as the most promising values of k, K_{max} and the depth of the covered area (the source region that we choose the preselected facility candidates from). Note that the customer-based neighbourhood method (*CNV3*) does not have such a flexibility as the value of k is fixed to 2 or 3, representing the number of critical points and also the source region is fixed being defined by the largest circle.

The learning consists of two stages. In the first stage, we record some information about the behaviour of the facility-based VNS. This is performed during a certain time period (say for instance 25% of the total time). The information that we are interested in includes the use of the k^{th} neighbourhood, the value of K_{max} and the depth of the covered area of the neighbourhood (the region that contains the preselected facility candidates). The second phase uses the information obtained to guide the search when using the facility-based VNS. Since VNS2(*FNV4*) is found to be the best performer, the learning process is carried out using this variant only.

3.5.1 Phase I: Learning process

In this phase, we gather the information mentioned above.

a) Depth of the covered area of the neighbourhoods (levels of the covering circle)

As the chosen facility is found dynamically, the levels used of the covering circle are identified whenever the solution improves. If there is an improvement at a given level, the frequency of using such a level will be increased by one.

b) Determination of the value of (k)

We also record the number of times the solution is improved in a given neighbourhood, say *k*. Furthermore, as part of the process we also identify the minimum and the maximum *k* values where the latter will define K_{max}.

3.5.2 Phase II: Using the information from phase 1

The information that is recorded in the first phase is then used to guide the search in VNS2(*FN4*). Two schemes are explored:

a) The range (min, max)

As the size of the covering circle is dynamic, we would like to determine the minimum and the maximum levels that have achieved improvement (i.e., the smallest and the largest covering circle that had been recorded). The same idea is also applied to fix the range for the value of (*k*), i.e. [*a, b*] where both *a* and *b* represent the smallest and the largest *k* respectively. Note that in the classical VNS, $a = 1$ and $b=K_{max}$. However, in some cases, it was observed that the number of these levels (k') and *k* can be further away from their respective means than what is deemed reasonable. Here, we consider those that lie beyond the mean + 2standard deviations as outliers and hence these are excluded from our analysis.

To illustrate this idea, we ran VNS2(*FNV4*) for 25% of the total time for 1000 runs of the multi-start algorithm and recorded the number of facilities that were around the largest circle and the *k* values whenever there was an improvement in the solution. As an example Table 3.8 shows that the solution improved 9 times during this period.

Table 3.8: Information recorded by applying existing data (n=439 TSP-Lib) with p=100

# Times improved solution	# candidate facilities around the largest circle	k value
1	10	1
2	14	1
3	27	1
4	43	2
5	35	3
6	24	4
7	10	4
8	21	6
9	24	3

In this case, we can conclude that after excluding the outliers, the range of the levels of the covering circle k' (the number of the candidate facilities which were around the largest circle) was (12 - 34) and the range of k was (1 - 4), see Table 3.9 for detailed results where the range was given by $[\mu - \sigma, \mu + \sigma]$

Table 3.9: The use of the range of the levels of the covering circle (k') and the number of neighbourhoods (k)

# Times improved solution	# candidate facilities around the largest circle	k value
1	10	1
2	14	1
3	27	1
4	43	2
5	35	3
6	24	4
7	10	4
8	21	6
9	24	3
Min	10	1
Max	43	6
Mean (μ)	23.111	2.778
Std. Dev (σ)	11.096	1.716
Range ($\mu - \sigma, \mu + \sigma$)	**(12.015, 34.207)**	**(1.062, 4.494)**

A preliminary study showed that this method has two weaknesses: (i) there is a possibility that some levels within the range did not improve the solution leading to a waste of time in exploring these levels, and (ii) the probabilities of using each level is considered to be the same, meaning that all levels are equally important. It was however observed that

some levels improved the solution several times while others only a few times or none. These two weaknesses also occur in determining the *k* values. The next scheme attempts to overcome these two weak points.

b) The frequency of occurrence

In this case, we take the information (say the levels of the covering circle) that has been recorded, and compute the probabilities of occurrence of each level, based on the number of times a solution is improved. These probabilities are then used to choose the covering circle (the level that contains the preselected facility candidates). In other words, the higher the probability of a given level or neighbourhood is, the higher the chance that such level or neighbourhood will be chosen. Figure 3.18 illustrates how such a scheme can be used.

The idea is to randomly choose $\alpha \in (0,1)$ uniformly and compute $\widehat{L} = F^{-1}(\alpha)$ with $F(L) = \sum_{t=1}^{L}(t)$ where P(t) refers to the probability of choosing the t^{th} level (t=1,..,p) or the t^{th} neighbourhood $(t = 1,..., K_{\max})$.

This technique is also referred to as the inverse method.

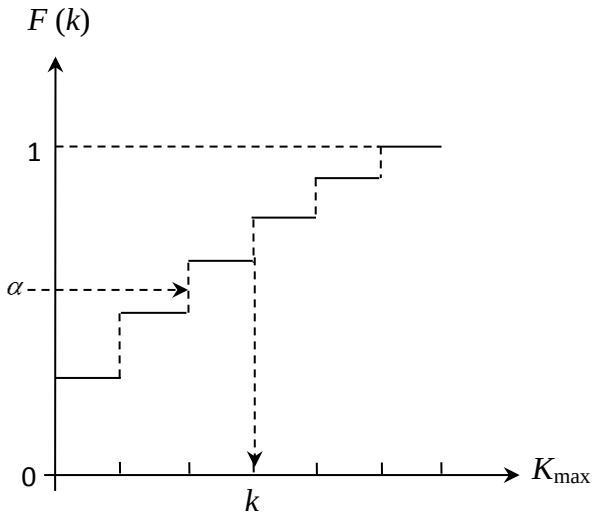


Figure 3.18: Selection of k using the frequency of occurrence

Based on the information that has been recorded in Table 3.9, the probability P(t) where t refers to the number facilities at that level and the value of k were computed in Table 3.10.

Table 3.10: Using the frequency of occurrence of the covering circle radii and the number of neighbourhoods (k)

# candidate facilities	P(candidate facilities)	Neighbourhood (k)	P(k)
10	2/9	1	3/9 = 1/3
14	1/9	2	1/9
21	1/9	3	2/9
24	2/9	4	2/9
27	1/9	6	1/9
35	1/9	-	-
43	1/9	-	-
$\sum$	1	-	1

This method was more adaptive as both the levels and the values of k were pseudo-randomly selected.

A preliminary experiment using both schemes on the same TSP data set with n= 439 and p varying from 10 to 100 in steps of 10 was given in Table 3.11. The results based on 10 runs showed that applying the frequency-based scheme was more efficient than the range-based.

Table 3.11: Deviation (%) of the average and the best results for VNS2(*FNV4*) using the range and the frequency of occurrence (10 random runs)

n = 439 TSP-Lib	The optimal solutions	Range				Frequency of occurrence			
p	(Z)	Deviation Average Results	Deviation Best Results	ST DEV	CPU * Time (secs)	Deviation Average Results	Deviation Best Results	ST DEV	CPU * Time (secs)
10	1716.5099	0	0	0	2.44	0	0	0	1.93
20	1029.7148	0	0	0	9.38	0	0	0	7.06
30	739.19297	0.29	0	6.69	13.15	0	0	0	9.42
40	580.00539	1.42	0	12.55	23.18	0.45	0	8.33	40.18
50	468.54162	3.09	0	12.38	86.68	2.69	0.94	4.04	48.63
60	400.19527	4.20	1.98	7.37	84.25	2.92	0.85	6.17	77.33
70	357.94553	1.58	1.23	3.28	69.79	1.42	1.27	1.62	65.39
80	312.5	6.73	2.45	11.74	74.22	5.38	1.98	4.27	100.99
90	280.90256	3.28	2.35	4.65	84.20	2.83	1.69	2.25	84.89
100	256.68019	5.95	3.54	5.95	92.11	3.90	1.30	4	115.02
Average	614.21882	2.65	1.16	6.46	**53.94**	**1.96**	**0.80**	**3.07**	55.08

*: *CPU time when the best solution was found.*

For instance, the overall average deviations for the best results were 0.80% and 1.16%, with the average results were 1.96% and 2.65%. The ST Deviation values of 3.07 and 6.46 of schemes 2 and 1 respectively, also confirm that the frequency-based scheme was more reliable especially for large values of p (eg; $p \geq 30$).

3.6 Computational results of customer-based vs facility-based with and without learning

In this section, our enhancements (customer-based neighbourhood (*CNV3*) and the facility-based VNS2(*FNV4*) with and without learning) were used to test the following existing data sets (n=439, 575, 783, 1002 and 1323 TSP-Lib) with various values of p (p=10 to 100 with an increment of 10). For n=439, we compared the computational results of our VNS based approaches to the optimal solutions provided by Chen and Chen (2009). For the other larger data set no optimal solutions are available. For our stopping criterion we performed the following experiment using a multi-start approach where we recorded the iteration number where the best solution was found after 1000 successive iterations without improvement. The detailed results were given in Appendix *C2* where the average and the maximum CPU times were recorded. In our study we therefore relate our stopping criterion to the CPU times corresponding to 10,000 iterations of the multi-start approach. This value was chosen as it was the smallest value in 1000s to cover all these maximum values, see Appendix *C2* for details.

3.6.1 Comparison against optimal results (small data set)

The TSP date set with n = 439 and various values of p was used for testing. These are the largest instances in the literature where optimal solutions were reported, see Chen and Chen (2009).

For simplicity and ease of repeatability, the initial solution in our VNS-based heuristics was taken as the solution of the multi-start algorithm with 100 runs. In Table 3.12, the results for VNS(*CNV3*) and VNS2(*FNV4*) with and without learning were reported. Our experiments showed that both VNS heuristics (*CNV3* and *FNV4*) produced better results than the multi-start heuristic. In brief, the performance of VNS(*CNV3*) was slightly inferior to the VNS2(*FNV4*) without learning as the overall average deviation values from the optimal solutions were 0.43% and 0.36% respectively. It can be seen that VNS2(*FNV4*) with learning (with memory) was more effective, as the overall deviation has been reduced to 0.23% besides the algorithm was able to find the optimal solution 5 times (i.e., when $p \leq 40$ and $p = 70$).

The optimal solutions were found by Chen and Chen (2009) who used a relaxation method based on solving a succession of small sub-problems. Their idea is to start solving a small sub-problem and check whether or not all points are covered. If it is true the optimal

solution is also optimal for the original problem, otherwise *k* demand points are added and the problem is solved again. This is repeated until the optimal solution is found for the entire problem. However, the new relaxation algorithms, though very interesting, have the drawback of not being able to guarantee in advance the best value of *k* that needs to be used besides the lack of identifying the initial sub-problems to start with. The authors reported those values of *k* that were found promising which can be difficult to reproduce when solving a new instance.

Table 3.12: Deviation (%) from the optimal solution of VNS2(*FNV4*) (with and without learning) and VNS(*CNV3*)

n	*p*	The optimal solutions	Multi-Start for 10,000 iterations	Neighbourhood Customer-based VNS(*CNV3*)	Neighbourhood Facility-based VNS2(*FNV4*)	
					No Learning	With Learning
		Z	Deviation	Deviation	Deviation	Deviation
439	10	1716.5099	2.02	**0**	**0**	**0**
	20	1029.7148	11.42	**0**	**0**	**0**
	30	739.19297	31.90	**0**	**0**	**0**
	40	580.00539	18.06	**0**	**0**	**0**
	50	468.54162	29.41	0.67	**0.28**	0.67
	60	400.19527	42.45	**0.35**	0.85	**0.35**
	70	357.94553	59.27	1.27	**0**	**0**
	80	312.5	30.70	1.20	1.20	**0.02**
	90	280.90256	25.93	**0.40**	**0.40**	**0.40**
	100	256.68019	27.46	**0. 40**	0.90	0.90
	Average	614.21882	27.86	0.43 (4)	0.36 **(5)**	**0.23 (5)**

( ): *The number of cases when the optimal solution is obtained.*
Bold: *The best solutions found.*

3.6.2 Results on larger data sets (no known optimal results)

Four larger existing datasets (*n*= 575, 783, 1002 and 1323 TSP-Lib) were used to assess the performance of our enhancements, see Table 3.13. As no optimal solution is available for these cases, we computed the deviation from the best solution as Deviation (%) = $\frac{(Z_H - Z_{best})}{Z_{best}}.100$ with Z_H denotes the Z value found by heuristic 'H' and Z_{best} refers to the best value of Z found by the heuristics. Here, we also used, as our initial solution, the solution of the multi-start algorithm with 100 runs.

In general, Table 3.13 shows that the performance of VNS2(*FNV4*) with learning outperformed all the others, yielded 20 best solutions. The *CN*-based approach VNS(*CNV3*)

produced the best solution 13 times while VNS2(*FN4*) without learning obtained 7 only out of 40 times.

Table 3.13: Deviation (%) of VNS2(*FNV4*) (with and without learning), VNS(*CNV3*), Multi-Start algorithm

n	*p*	*Z*	Deviation (%)			
		Overall best solutions	Multi Start (10,000 Iterations)	VNS(*CNV3*)	VNS2(*FNV4*)	
					No Learning	With Learning
575	10	67.926	1.91	1	1	**0**
	20	45.622	3.10	**0**	0.75	**0**
	30	35.556	9.05	**0**	**0**	0.16
	40	30.265	14.51	1.65	1.26	**0**
	50	26.173	17.84	0.37	2.40	1.13
	60	23.622	18.71	2.52	**0**	0.29
	70	21.059	14.76	2.12	1.77	**0**
	80	19.558	24.96	0.17	1.88	1.49
	90	17.923	23.57	0.81	2.37	**0**
	100	16.621	28.51	0.54	0.46	0.47
	Average	30.433	15.69	0.92	1.19	**0.35**
783	10	79.313	**0**	**0**	**0**	**0**
	20	53.461	2.713	0.43	0.69	**0**
	30	42.395	11.84	**0**	2.06	0.49
	40	35.962	10.75	1.59	**0**	0.41
	50	31.409	15.17	0.19	0.87	**0**
	60	28.053	18.19	**0**	0.04	1.10
	70	25.446	20.89	**0**	1.57	0.69
	80	23.560	22.67	0.85	0.06	**0**
	90	21.710	24.71	1.57	3.67	**0**
	100	20.334	26.01	1.09	2.03	**0**
	Average	36.164	15.29	0.57	1.10	**0.27**
1002	10	2389.360	0.89	**0**	**0**	**0**
	20	1609.540	4.66	**0**	1.29	1.29
	30	1231.360	8.42	0.11	2.02	**0**
	40	1030.400	17.06	1.30	1.35	**0**
	50	906.228	16.39	**0**	1.14	0.19
	60	801.474	21.13	0.15	2.20	**0**
	70	727.154	17.70	0.98	1.29	**0**
	80	664.798	22.03	1.72	2.30	1.05
	90	604.494	28.27	**0**	1.73	0.75
	100	559.017	29.43	2.08	3.73	2.08
	Average	1052.383	16.60	0.63	1.71	**0.54**
1323	10	2897.490	0.33	0.24	0.07	0.07
	20	1886.820	4.41	**0**	**0**	**0**
	30	1466.970	8.29	1.62	2.67	0.98
	40	1236.380	12.41	**0**	0.34	1.21
	50	1060.820	15.99	**0**	1.46	0.42
	60	941.870	12.66	1.23	2.19	**0**
	70	844.967	19.38	0.93	1.62	**0**
	80	774.764	15.34	1.09	2.65	**0**
	90	720.625	24	0.66	**0**	2.12
	100	662.936	28.63	2.24	1.66	5.13
	Average	1249.364	14.14	0.80	1.27	0.99
Overall Average		592.086	15.43	0.73	1.32	**0.54**
# Best			(1)	(13)	(7)	**(20)**

( ): *The number of cases when the optimal solution was obtained.*
Bold: *The best solutions found.*

It can also be observed that the multi-start algorithm (10,000 runs) achieved the best solution only once. In addition, the average deviation values also confirm that the performance of VNS2(*FNV4*) with learning always yielded relatively better results than those of the other enhancements, with an overall average deviation of 0.54%. These compare favourably with 0.73% and 1.32% for VNS(*CNV3*) and VNS2(*FNV4*) without learning respectively.

In brief, we can confirm that the performance of VNS(*CNV3*) was better than VNS2(*FNV4*) without learning, but the incorporation of learning into the search has made VNS2(*FNV4*) to be the best performer.

3.6.3 Time performance

A comparison between the average total CPU time of the Multi-Start algorithm (10,000 iterations) and the average CPU time when the best continuous solution was found as well as Chen and Chen's results (when it is available) was presented in Table 3.14. It is worth noting that the recording of when the best solution was obtained could be useful in designing a more advanced stopping rule. For instance this can show that using a stopping criterion where the search terminates after a certain number of runs (or time) without improvement. To achieve this, we recorded the CPU time when the best solution was found by a given heuristic as T_H and computed the deviation from the CPU time required for 10,000 iterations of the multi-start algorithm which we refer to as $T_{M.S}$. This was computed as follows:

$$\text{Deviation (\%)} = \frac{(T_H - T_{M.S})}{T_{M.S}}.100$$

To provide a fair comparison in terms of CPU, we used the following transformation as given by Dongarra (2013) with $T_2 = T_1 \frac{n_1}{n_2}$, where T_1 represents the reported time in Machine 1 and T_2 the estimated time in Machine 2. n_1 and n_2 refer to the number of Mflops in machines 1 and 2, respectively. For more information, see http://www.roylongbottom.org.uk. As the computer used by Chen and Chen (2009) cannot be easily identified for the number of Mflops, we provided an approximate time using a slightly slower but similar computer namely a PC Intel Pentium 4 (3.06 GHz), 2 GB of main memory.

For each value of *n* we ran the methods for *p*= 10 to 100 with an increment of 10 and recorded the average results. The details were given in Appendix *C2* but the overall average deviation of the CPU time when the best solution was found was shown in Table 3.14.

Table 3.14: Average CPU time of the Multi-Start algorithm (for *p*=10 to 100 in increment of 10), Deviation (%) of CPU time for VNS2(*FNV4*) (with and without learning) and VNS(*CNV3*)

n	Average total CPU time (10,000 iterations) (s)	Deviation (%)				
		VNS(*CNV3*) (Best CPU Time)*	VNS2(*FNV4*) (Best CPU Time)*		Chen and Chen's results (Continuous Solutions)	
			No Learning	With Learning	Improved relaxation (*k*=7)	Binary relaxation (*k*=6)
439	1497.56	-81.73	-73.21	-74.64	-88.93	-98.76
575	1681.81	-55.90	-47.52	-36.91	N/A	N/A
783	2762.45	-39.85	-39.65	-48.84	N/A	N/A
1002	4398.09	-45.43	-59.28	-57.98	N/A	N/A
1323	5662.98	-50.79	-33.67	-48.44	N/A	N/A
Average	3200.58	-54.74	-50.67	-53.36	N/A	N/A

*: *CPU time when the best solution is found.*
k:is the best recorded value in Chen and Chen (2009).

Table 3.14 showed that the overall deviations of CPU time when the best solution was found increased with *n* for all the algorithms. For instance, in VNS(*CNV3*), the overall deviations were found to be -81.73% and -50.79 % for *n* = 439 and 1323 respectively.

In general, it can be noted that applying VNS(*CNV3*) and VNS2(*FNV4*) required around 50% of the time required by the multi start algorithm.

3.7 Summary

In this chapter, we first presented a basic variable neighbourhood search algorithm (VNS) using two types of neighbourhoods namely customer-based VNS(*CN*) and facility-based VNS(*FN*). Furthermore, we proposed three enhancements for VNS(*CN*) and four for VNS2(*FN*). Two modifications are also introduced to our local search (Cooper's approach) to make it more efficient. Schemes based on identifying neighbourhoods around the critical points such as specific rules for eliminating circles with a few points etc, proved to be useful. In addition, the effect of learning when used within VNS2(*FNV4*) is explored and proved to be useful in improving the solutions.

For n = 439 where optimal solutions are reported, the computational results show that the variant VNS2(*FNV4*) without learning overcame the other enhancements of the neighbourhood customer-based, where the overall deviation (%) in the objective function values is 0.36%. For the customer-based the variant VNS(*CNV3*) produced an overall deviation of 0.43%. It was found that VNS2(*FNV4*) with memory (with learning) is more effective, as the overall deviation has been reduced to 0.23%. Furthermore, this variant is able to find the optimal solution 5 times out of 10. For larger datasets (n= 575, 783, 1002 and 1323), with no known optimal solution or best known, this method also outperforms the others by obtaining the best solution 20 times out of 40.

In the next chapter we will explore another meta-heuristic that uses perturbation ideas to address the same problem.

Chapter 4

Perturbation-Based Heuristics

4.1 Introduction

In this chapter, a brief review of perturbation-based heuristics is first given followed by the three types of moves that are adapted for the continuous *p*-centre problem. The original perturbation algorithm, which refers to the gradual perturbation "GRADPERT" is revisited in this study by allowing flexibility in the amount of perturbation leading to two new implementation namely "STRONGPERT-V1 and STRONGPERT-V2". Powerful enhancements are then designed and embedded in GRADPERT and STRONGPERT-V2. The incorporation of learning within the search is shown to be effective by making the search more adaptive. The computational experiments show the high quality results found by these enhancements when compared to the original algorithm and the VNS meta-heuristic.

4.2 A perturbation-based heuristic

This approach guides the search by introducing some perturbations or noises into the problem. For the *p*-centre problem these can be achieved by allowing the number of facilities of a solution to go over and under the required number of facilities (*p*), by a certain value (*q*). In other words, the solution is allowed to be infeasible in terms of the number of open facilities. An initial solution of the p-centre problem is first found, then the number of open facilities is allowed to increase to $\overline{p}\,(\overline{p} = p+q)$ by adding q facilities to the current solution. The removal of q facilities is then performed to reach a solution with p facilities where an intensification of the search is activated. The removal of facilities continues till the problem with $\overline{\overline{p}}$ facilities is reached $(\overline{\overline{p}} = p-q)$. At this stage the addition of q facilities is performed to get a feasible solution with p open facilities where intensification is activated again. We refer to this up and down shifting as one cycle of the perturbation procedure which is then repeated several times until one of the following two stopping criteria is met

whichever comes first. These include the maximum number of cycles without improvement ($NCycle_{\max}$) or the total *CPU* time reaches the maximum time allowed ($CPU_{\max}$).

The idea of moving between feasible and infeasible regions acts as a filtering process where the most attractive facilities have the tendency to remain in the promising set. Salhi (1997) proposed this heuristic for a class of large uncapacitated location problems including the *p*-median problem with interesting results. Hanafi and Freville (1998) also adapted a similar approach for solving a class of knapsack problems, whereas Zainuddin and Salhi (2007) modified this methodology to solve the capacitated multisource Weber problem. In the perturbation-based heuristic, there are three types of moves, namely the add, the swap, and the drop moves. In the add move, q open facilities are added and in the drop move q open facilities are removed from the current solution. The swap move is applied when the number of open facilities is p. In other words, when the number of facilities is p, a form of intensification based on the swap move is applied whereas when the solution has $p \pm s$, s=1,...,q facilities a diversification is used instead.

It is worth noting that the idea of perturbation shares some similarities, especially when going from p to $p - q$ then up to p, to the large neighbourhood search proposed by Shaw (1998) and which proved to be successful when applied to a class of routing problems by Pisinger and Ropke (2010). More information and references for large neighbourhood search can be found in Ahuja *et al.* (2002).

In this study we revisit this perturbation type meta-heuristic by introducing flexibility in the level of perturbation using a variable value of q that is adaptively determined instead of considering a constant value throughout the search as usually used in the literature. Besides, the moves adopted for this problem such as the swap, the add and the drop moves as well as the way the optimal location is found within each cluster are also tailored to the *p*-centre problem. In this perturbation approach, two types of local search are examined; When the number of facilities of a solution is $p \pm s, s = 1,...,q$, we apply the locate-allocate procedure. We refer to this as the local search of type 1 "LS1". When the number of facilities is p we use a combined local search "LS2" made up of LS1 and the swap-based neighbourhood that will be described later.

The perturbation-based heuristic includes three types of moves, which are as follows: (i) the add move where q facilities are added into the current solution, (ii) the drop move where q open facilities are removed from the current solution and (iii) the swap move when the number of open facilities is p. This is performed by swapping one of the open facility location with a location in the continuous space either random or using forms of guidance.

A brief explanation of the three types of moves

In this subsection, the three moves that are used in our perturbation-based heuristic are presented alongside some basic enhancements.

The drop move

The strategy that we adopt is to remove q facilities one by one followed by LS1, which is a "locate-allocate" procedure similar to that of Cooper (1964) as explained in subsection 3.2.1 of chapter 3. This process is applied when the number of open facilities is p and going down to $p-q$ or starting from $p+q$ and going down to p. Here, the facility chosen for removal is the one whose removal increases the objective function the least where the local search LS1 is then applied to find the new solution with one facility less. This procedure is repeated until q facilities are removed.

The add move

Here, q facilities are inserted when the number of open facilities is p with the aim to go over the required number of facilities to $p + q$ (*infeasible case*). Similarly this is also applied when the number of open facilities reaches $p-q$, when we start adding facility one by one till we get to p. The way the new added facility is performed will be described in subsequent sections when LS1 is also applied at each new solution with $p\pm$s facilities (s=1,...,q).

The swap move (LS2)

The idea of the covering circle which was successfully applied for the facility-based VNS2(*FNV2*) is also adapted in this swap move. When the number of open facilities reaches p, we relocate randomly a location of one facility in the continuous space of the current covering circle CC_k.

For example only for illustration all the (CC_k) k=1,…8 of an *8*-centre problem are shown in Figure 4.1. The formal definition of CC_k can be found in subsection 3.3.2 of chapter 3.

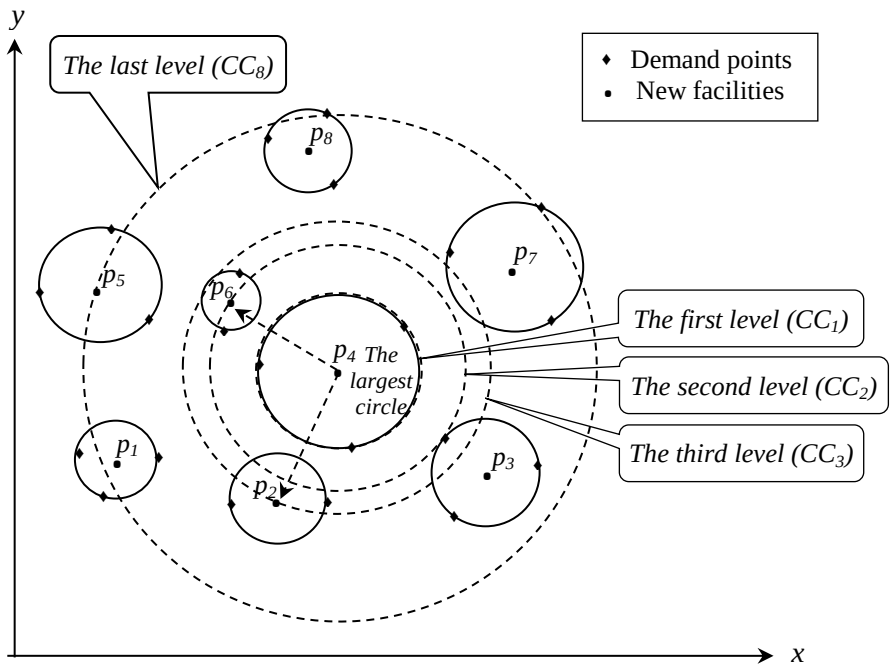


Figure 4.1: An example of the levels of covering circles that are dynamically increasing from the source region of a 8-centre problem (i.e., largest circle)

In brief, the procedure works as follows: we start from the first level (k = 1) of the covering circle CC_1 (the largest circle) by dropping the facility sited at the centre of the largest circle and inserting a facility randomly in CC_1. If the solution is not improved, when applying LS1, we move to CC_2 which contains two facilities namely those sited at the centres of the largest circle and the nearest facility to it. One of these two candidate facilities is then randomly selected for dropping and inserting a facility randomly in CC_2 which is then followed by LS1. If the new solution is improved we revert back to level 1, where the largest circle is identified again. Note that this is not necessarily the previous largest circle. Its corresponding covering circle CC_1 is then defined and the process is repeated. If the solution is not improved we extend the search by exploring the next level. This process continues until the last level say $l_{\max}$, which includes all the facilities, is reached. From that point we then start to reduce gradually the level of the covering circle until we get to level one. The swapping process is performed until no improvement is found after $K_{\max}$ successive times (here we set

$K_{\max} = \lceil \sqrt{p} \rceil$). Note that at this point, we record the current level, say $\hat{l}$, and the direction whether we are in the process of increasing the level (Flag = 1) or decreasing the level (Flag = -1). This is important as this information is used when we reach *p* again in subsequent iterations as the search continues from the next level based on whichever level is reached at this iteration. In other words, if Flag=1 we set $l = l + 1$, else we set $l = l - 1$ while we continue using the direction as defined by Flag. Initially Flag is set to 1 as the search starts from level 1 which is based on CC_1.

We propose two types of perturbation, which are based on an adaptation of the perturbation originally given by Salhi (1997). We refer to the first one as the gradual perturbation "GRADPERT" and the second as the strong perturbation "STRONGPERT". These two strategies differ in the way the *q* facilities are added within the search. These are described in the next two sections.

4.3 The original perturbation (GRADPERT)

Here, the new *q* facilities are added randomly one at a time in CC_1 (area encompassed by the largest circle) where "LS1" is activated in each of the *q* steps. A similar process is also used in the drop move except that the removal is not performed randomly. Note that in Salhi (1997) the added facility is inserted based on the largest cost saving over the potential facility sites as the problem was not a continuous but a discrete type location problem. For example, if *q* = 2, the first facility is inserted randomly in CC_1, then LS1 is applied and the new largest circle is found again and CC_1 defined. The second facility is then located in the new CC_1 where the number of open facilities becomes *p*+2. Similarly in the drop move, the facility that increases the objective function the least is dropped from *p*+2 to *p*+1, followed by LS1. A second facility is then chosen for removal by using the same procedure to reach *p*, see Figure 4.2. Here, when the number of facilities is *p*, the swap move (LS2) is activated using a procedure PROC_LS2 which will be described in the next subsection.

A full cycle of the perturbation which contains the three types of moves is defined as follows: starting from a solution with *p* facilities, *q* adds are followed by *q* drops, a swap is activated followed by *q* drops then *q* adds and finally a swap. See Figure 4.2 for an illustration

where a full cycle is represented. It is worth noting that there are two strategies that can be applied which differ in the choice of the solution used at the start in each cycle of the perturbation. These are described in the next two subsections.

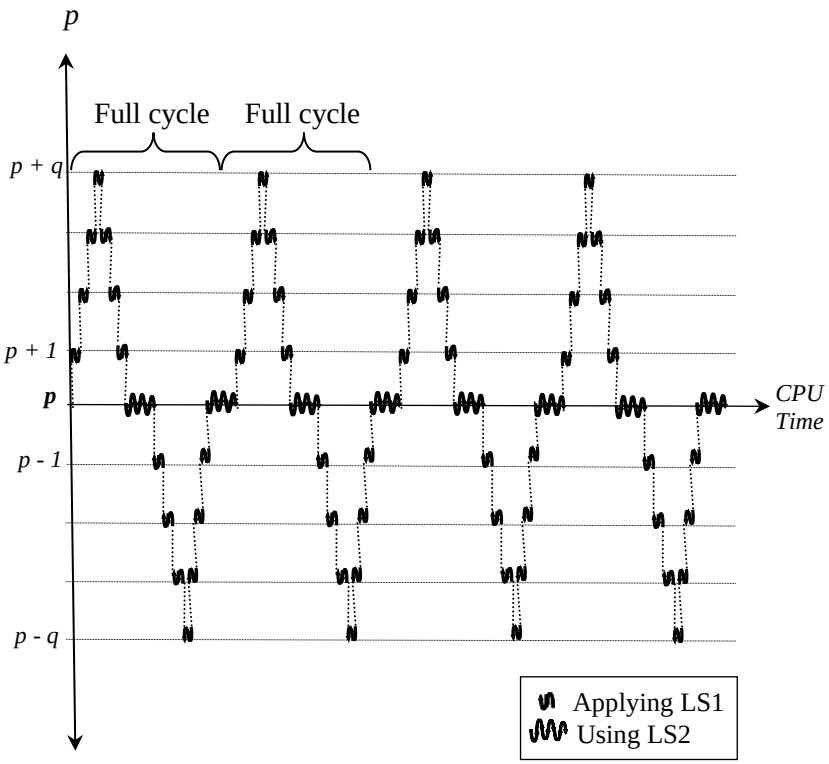


Figure 4.2: Gradual perturbation GRADPERT (*q is fixed*)

4.3.1 The first strategy (using the current solution)

In this strategy, the current feasible solution (the last feasible solution that has been obtained) is used as the initial solution at the start of each cycle of the perturbation.
Figure 4.2 illustrates the mechanism of "GRADPERT" whose steps are given in Figure 4.3.

Explanation of some of the steps

Step 1: The initial solution is generated randomly by choosing *p* fixed points. This solution is then improved using LS1, though other schemes could also be used such as the best solution of a multi-start procedure (say, with 100 runs) or the optimal solution of the vertex *p*-centre problem. The former will be tested in the computational results section. Note that we consider

the CPU_{max} as the only condition as this is based on the CPU time recorded by 10,000 multi-start. In general, we could also use $Cycle_{max}$ as another stopping condition.

Step 0: Set q, K_{max}, l_{max} and CPU_{max} and let $\bar{p} = p$, $\hat{l}$=1and *Flag*=1.

Step 1: Generate an initial feasible solution (X) and compute the objective function value $Z(X)$.

Set $X_{best} = X$ and $Z_{best} = Z(X)$.

Step 2:

Step 2*a*: Perturb the recent solution (X) by adding randomly one facility in CC_1, apply LS1 to find the new X and set $\bar{p} = \bar{p} + 1$ ***"Perturbing by adding"***

Step 2*b*: If $\bar{p} = p$, apply LS2 using PROC_LS2($\hat{l}$, l_{max} , K_{max}, $\hat{X}$, *Flag*) ***"Swapping Step"***

If $Z(\hat{X}) < Z_{best}$ then set $X_{best} = \hat{X}$ and $Z_{best} = Z(\hat{X})$;

Set $X = X_{best}$

Step 2*c*: If $\bar{p} < p + q$ go to *Step* 2*a*, else go to *Step* 3.

Step 3:

Step 3*a*: Perturb the recent solution (X) by dropping the facility yielding the least extra cost, apply LS1 to find the new X and set $\bar{p} = \bar{p} - 1$ ***"Perturbing by removing"***

Step 3*b*: If $\bar{p} = p$, apply LS2 using PROC_LS2 ($\hat{l}$, l_{max}, K_{max}, $\hat{X}$, *Flag*). ***"Swapping Step"***

If $Z(\hat{X}) < Z_{best}$ set $X_{best} = \hat{X}$ and $Z_{best} = Z(\hat{X})$;

Set $X = X_{best}$

Step 3*c*: If $\bar{p} > p - q$ go to *Step* 3*a*, else go to *Step* 4.

Step 4: If CPU time > CPU_{max} record X_{best}, Z_{best} and stop, else go to *Step* 2.

Figure 4.3: A gradual perturbation algorithm (GRADPERT)

Steps 2a and 3a (Applying the local search (LS1))

This local search is the same one that is used in subsection 3.2.1.

Steps 2b and 3b (the use of LS2)

When a solution with p facilities is obtained, an intensification is activated using the procedure PROC_LS2 which will be described next. Here, a swapping process is used where one facility is chosen randomly from the covering circle (level $l = \hat{l}$) and then relocated randomly in the same covering circle. Note that initially $\hat{l} = 1$. The covering circle is a circle

with a dynamically increasing radius, given by the distance between the centre of the largest circle and its $(l)^{th}$ nearest facility, as shown in Figure 4.1. In other words, if the solution is not improved, the radius of the covering circle increases with l (l =1,...,l_{max}) until the last level (l_{max}) is reached. At this point, the level is reduced gradually until level one (l_1). However, if the solution is improved we return to l_1 again which is CC_1 in the current solution. This swapping process is repeated until K_{max} iterations are performed without improvement where we record the best solution $\hat{X}$, the current level reached $\hat{l}$ and the direction of the search (increasing or decreasing using Flag=1 or -1 respectively). The steps of this procedure, which we call PROC-LS2, are given in Figure 4.4.

PROC_LS2($\hat{l}$, l_{max}, K_{max}, $\hat{X}$, $Flag$)

(i) Let k=0 , X=X_{best} and $l = \hat{l}$

(ii) Generate a new feasible solution by swapping randomly one facility from X in level l, apply LS1 and set k=k+1. Let X' be the new solution.

(iii) If $Z(X') < Z(X)$, set $X = X'$, k=0, l=1, and go to step (ii)

(iv) If FLAG=1

If $l < l_{max}$ then set l=l+1, else FLAG= -1

Else

If l >1 then set l= l -1, else FLAG=+1

(v) If $k < K_{max}$, set k=k+1 and return to step (ii), otherwise set $\hat{X} = X$ and $\hat{l} = l$.

Figure 4.4: The PROC-LS2 procedure

4.3.2 The second strategy

In this strategy, the best feasible solution that has been obtained so far is used as the initial solution at the start of each cycle of the perturbation.

The steps for using this strategy in GRADPERT are similar to those of Figure 4.3 except that in *Step* 2*a* and *Step* 3*a* the best solution is perturbed instead of the current one. For clarity, these 2 steps are reproduced here.

Step 2*a*: Perturb the best solution (X_{best}) by adding randomly one facility in CC_1, apply LS1 to find the new X and set $\bar{p} = \bar{p}$ +1

Step 3*a*: Perturb the best solution (X_{best}) by dropping the facility that increases the objective function the least, apply LS1 to find the new X and set $\bar{p} = \bar{p} - 1$

4.3.3 Empirical computational results of the original perturbation

In this section, the existing data set (n=439 TSP-Lib) with p=10 to 100 was used to assess the performance of using the above two strategies. GRADPERT was run for 10 times, starting from the same random initial solution, which was the solution of the multi-start algorithm with 100 runs. Each run used a total CPU time $\frac{\text{CPU}_{\text{MS}}}{10}$ where CPU_{MS} was the CPU for the 10,000 runs as performed in the previous chapter.

From Table 4.1, it can be clearly observed that using the second strategy yielded better results than those found by the first one. This was shown in terms of both the best and the average solutions. We also recorded the CPU time when the best solution was found for information only as this demonstrates that other stopping criteria such as max cycle without improvement could be useful.

Table 4.1: Deviation (%) of the average and the best result from the optimal solution for the first and the second strategy

n= 439 TSP-Lib p	The optimal solutions (Z)	The first strategy (using the current solution)				The second strategy (using the best solution)			
		Deviation Average Results	Deviation Best Results	ST DEV	CPU Time*	Deviation Average Results	Deviation Best Results	ST DEV	CPU Time*
10	1716.5099	0	0	0	10.04	0	0	0	9.22
20	1029.7148	0	0	0	11.37	0	0	0	11.72
30	739.19297	0	0	0	20.95	0	0	0	17.42
40	580.00539	0	0	0	51.17	0	0	0	26.74
50	468.54162	2.75	2.16	1.94	76.96	1.49	0	4.52	113.35
60	400.19527	2.85	2.46	1.39	81.60	2.71	0.35	5.59	63.58
70	357.94553	1.40	1.27	0.77	74.58	1.28	1.23	0.30	101.58
80	312.5	5.51	3.62	3.52	101.09	4.36	1.20	4.13	139.13
90	280.90256	2.89	2.35	2.15	90.61	2.42	2.19	1.09	90.19
100	256.68019	5.59	3.54	3.90	68.14	3.87	3.54	1.04	134.48
Average	614.21882	2.10	1.54	**1.37**	58.65	**1.61**	**0.85**	1.67	70.74

*: *CPU time when the best solution was found.*

The average deviation values from the optimal for the best results were found to be 0.85% and 1.54% respectively with the average results of 1.61% and 2.10%. This level of performance was relatively high showing the power of perturbation. In addition, the second one achieved the optimal solution 5 times whereas the first obtained 4 out of 10 times.

However, the solutions of the latter were found to be less spread than the ones by the former as shown by their ST deviations of 1.37 vs 1.67.

4.4 The strong perturbation (STRONGPERT)

In this variant, when starting from a solution with p facilities, all the q facilities are added randomly in one step. Similarly, q facilities are also added randomly for a solution with $p-q$ facilities to reach p facilities in one step, see Figure 4.5. However, when we apply the drop move from $p+q$ to p and from p to $p-q$, we retain the same dropping process that was used in the gradual perturbation. We call this variant "STRONGPERT", short for the strong perturbation.

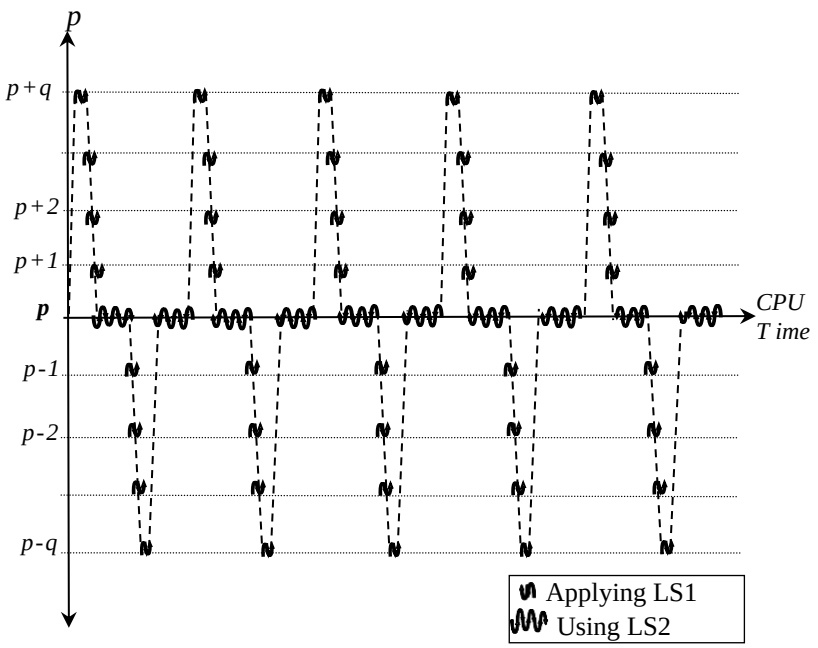


Figure 4.5: Strong perturbation STRONGPERT (*q is fixed*)

Since the second strategy yields relatively better solutions than the ones found by the first strategy, as shown in Table 4.1, this is the chosen strategy that we select in STRONGPERT. Here, we also introduce two schemes for adding the new q facilities which differ in the way these new facilities are located randomly in the continuous space. These two variants are explained in the next subsections.

4.4.1 STRONGPERT-V1

In this scheme, we will focus on the larger circles. The idea behind this enhancement is to insert randomly q facilities in one step in the areas covered by the q larger circles (1^{st} largest, the 2^{nd} largest, 3^{rd} largest,..., $(q)^{th}$ largest circle). To illustrate this idea, consider $q_{max} = \lceil \sqrt{p} \rceil$, with $p = 9$, (i.e., $q_{max} = 3$). In this case, three new facilities are located randomly in the areas covered by the first, the second and the third largest circles separately. Once this is performed the local search LS1 is then applied. Figure 4.6 illustrates these new facilities denoted by $\bar{p}_{10}, \bar{p}_{11}$ and $\bar{p}_{12}$.

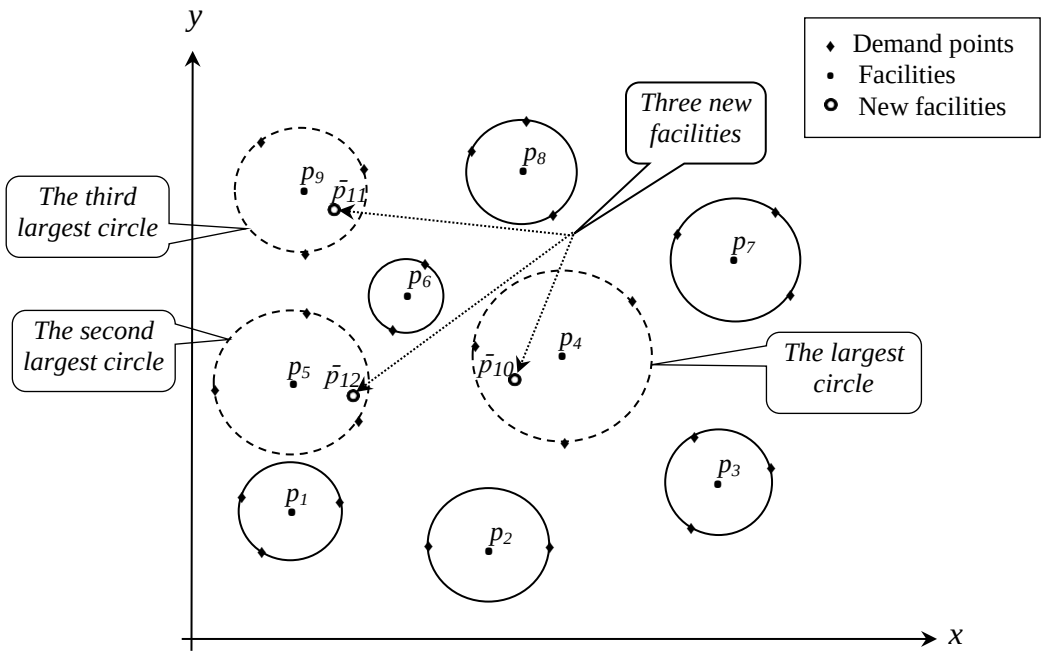


Figure 4.6: The first, the second and the third covering region (q_{max}=3) of a 9-centre location problem

In general, the steps of this algorithm (STRONGPERT-V1) are similar to those of the original algorithm (GRADPERT) of Figure 4.3 except that in *Step* 2*a* we add randomly q facilities in the areas covered by the q larger circles separately. For simplicity, *Step* 2*a* is given here.

Step 2*a*: Perturb the best solution (X_{best}) by adding randomly q facilities in the areas covered by the q larger circles separately then set $\bar{p} = \bar{p} + q$.

4.4.2 STRONGPERT-V2

When the solution of the *p*-centre location problem is not optimal, it can be noted that the facility serving the customers that are encompassed by the largest circle and at least one of the facilities that are around it are not in the right location. In this variant, which we call **"**STRONGPERT-V2" (short for the strong perturbation variant 2), we take this observation into account and introduce extra perturbations into the current solution (around the largest circle) in one step. Here, the idea of the covering circle is also used to introduce these extra perturbations. This is performed by adding randomly all the q facilities in CC_q where LS1 is then activated. For example, see Figure 4.7 when q = 3. Here, these three new facilities ($\bar{p}_{10}$, $\bar{p}_{11}$ and $\bar{p}_{12}$) will be located randomly in CC_3. The steps of this scheme (*STRONGPERT*-V2**)** are similar to the ones of GRADPERT of Figure 4.3 except that *Step* 2*a* is replaced as follows:

Step 2*a*: Perturb the best solution (X_{best}) by adding randomly q facilities in the continuous space encompassed by the covering circle CC_q then set then set $\bar{p} = \bar{p} + q$.

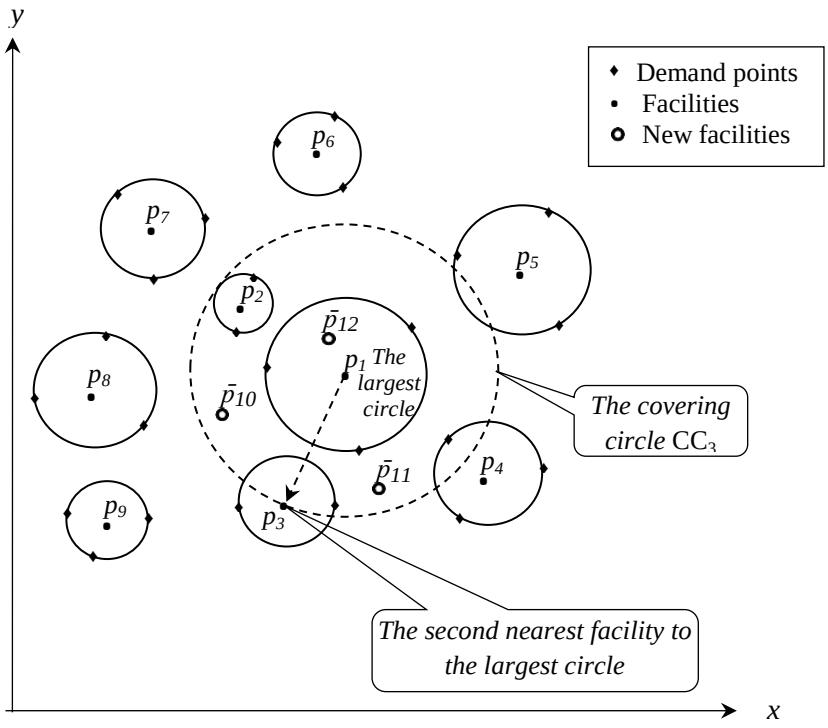


Figure 4.7: An example of the levels of covering circles that are dynamically increasing from the source region of a 9-centre location problem

4.4.3 Empirical results of STRONGPERT-V1 vs STRONGPERT-V2

In this section, computational results for both the STRONGPERT-V1 and the STRONGPERT-V2 were presented. The same data that were used to assess the performance of the original perturbation were also used here. Each algorithm was run for 10 times, using a total time for each run as $\frac{\text{CPU of 10,000 runs for MSALA}}{10}$.

Table 4.2 shows that, in terms of the best solution, STRONGPERT-V2 was found to be better and relatively faster than STRONGPERT-V1 with an average deviation of 0.70% compared to 1.01%. However, the latter yielded solutions which were less spread than the ones found by its equivalent STRONGPERT-V2 (see ST deviation of 2.18 vs 2.47). However, the average performance of STRONGPERT-V2 was slightly inferior to its counterpart STRONGPERT-V1 with an average deviation of 1.68% and 1.63% respectively. STRONGPERT-V2 will, given its encouraging promising performance, be explored further in subsequent sections.

Table 4.2: Deviation (%) of the average and the best result from the optimal solution for STRONGPERT-V1 and STRONGPERT-V2

n=439 TSP-Lib	The optimal solutions	The strong perturbation (STRONGPERT)							
		STRONGPERT-V1				STRONGPERT-V2			
p	(Z)	Deviation Average Results	Deviation Best Results	ST DEV	CPU Time*	Deviation Average Results	Deviation Best Results	ST DEV	CPU Time*
10	1716.5099	0	0	0	6.93	0	0	0	6.99
20	1029.7148	0	0	0	10.31	0	0	0	10.40
30	739.19297	0	0	0	15.18	0	0	0	18.99
40	580.00539	0.53	0	8.29	37.08	0.45	0	8.33	29.40
50	468.54162	1.73	0.47	3.64	107.48	1.80	0.67	3.42	68.94
60	400.19527	2.55	1.98	1.19	91.06	2.44	0.35	3.21	81.43
70	357.94553	1.32	1.23	0.37	78.61	1.25	1.23	0.08	83.83
80	312.5	4.84	2.45	5.11	90.64	5.17	1.20	6.15	85.03
90	280.90256	1.69	0.40	2.58	91.80	1.89	0.40	2.29	120.35
100	256.68019	3.67	3.54	0.62	129.81	3.80	3.19	1.18	114.63
Average	614.21882	**1.63**	1.01	2.18	65.89	1.68	**0.70**	2.47	62.00

*: *CPU time when the best solution was found.*

4.5 Perturbation-based enhancement

In both perturbations, the number of the new facilities (q) that are used to go over and under the required number of facilities (p) is fixed (say for instance $q = q_{max}$). However, in this enhancement the value of q is relaxed and made dynamic starting from q =1 to q_{max} with

an increment of 1. This enhancement is used in both perturbations where in GRADPERT, LS1 is applied at each solution with a number of facilities $p \pm$ s, s = 1,..., q_{max}, while in STRONGPERT, LS1 is applied at the adding phase when $q = q_{max}$ only. However, in both perturbations the local search LS2 is still used whenever a solution has p facilities.

4.5.1 Enhancement on GRADPERT (Enh 1)

This is formally defined as follows:

Let a cycle of size q be defined as cycle (q) representing the sequence of moves made up of q add moves followed by q drops moves then a call to "LS2", a drop of another q moves followed by q add moves and finally a call to LS2 again.

In GRADPERT, cycle (q_{max}) is used throughout the search several times whereas here we use cycle (q) instead with q =1,2,....., q_{max}. See Figure 4.8 for an illustration. The main steps of this enhancement, which we call "Enh 1", are similar to GRADPERT in Figure 4.3 except that in *Step* 0 we use q=1 and $q_{max} = \lceil \sqrt{p} \rceil$, and *Step* 3*c* is replaced by:

Step 3c:

If $\bar{p} > p - q$, go to *Step* 3*a* (*it is unchanged*),
else
if $q < q_{max}$, set $q = q+1$, else set q =1;
go to *Step* 4.

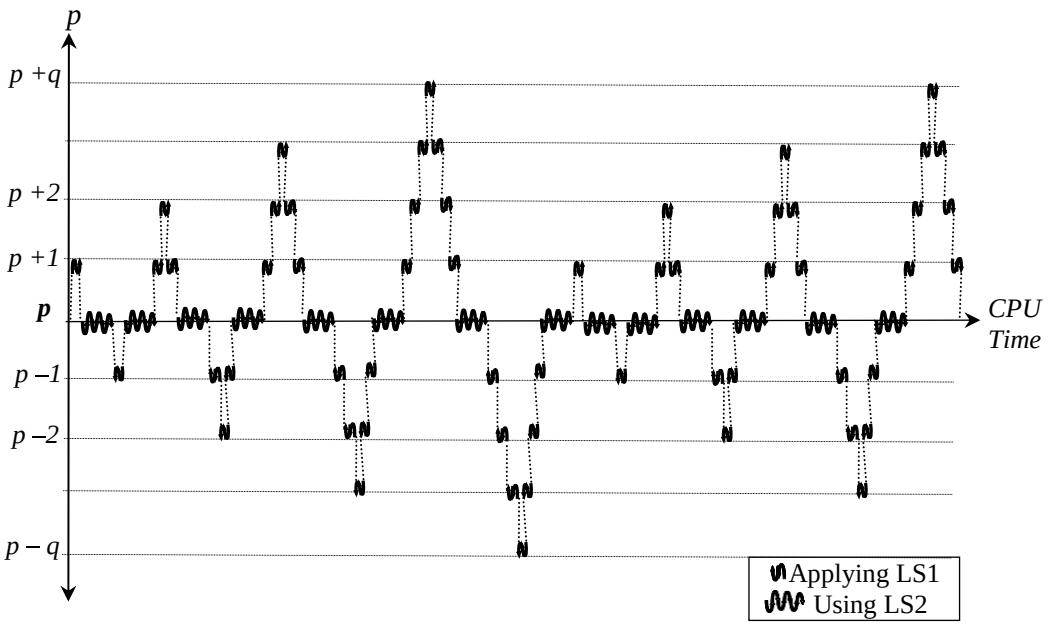


Figure 4.8: The Gradual perturbation GRADPERT (*with dynamic q*)

4.5.2 Enhancement on STRONGPERT-V2 (Enh 2)

This is a hybrid of Enh 1 where the value of q is dynamic (q =1,..., $q_{\max}$) instead of being fixed, and STRONGPERT where the q new facilities are inserted randomly in the area of the covering circle (i.e., CC_q) in one step. We call this enhancement Enh 2. Here, we start to add one facility randomly (q=1) into the first level of the covering circle (i.e., the first level referring to CC_1). In case of ($q=2$), two new facilities are added randomly in the covering circle CC_2 (the second level). This radius of the covering circle continues to increase with q until the last level is reached (i.e., $CC_{q_{\max}}$). Recall that the last level is the covering circle with its radius as the distance between its centre (centre of C_1) and its $(q_{\max})^{th}$ nearest facility.

Note: It is worth highlighting that the levels of the covering circle are determined by the value of q. The removal process is similar to the one used in STRONGPERT-V2, where q facilities are removed one by one followed by LS1 at each of the single moves, see Figure 4.9 for an illustration.

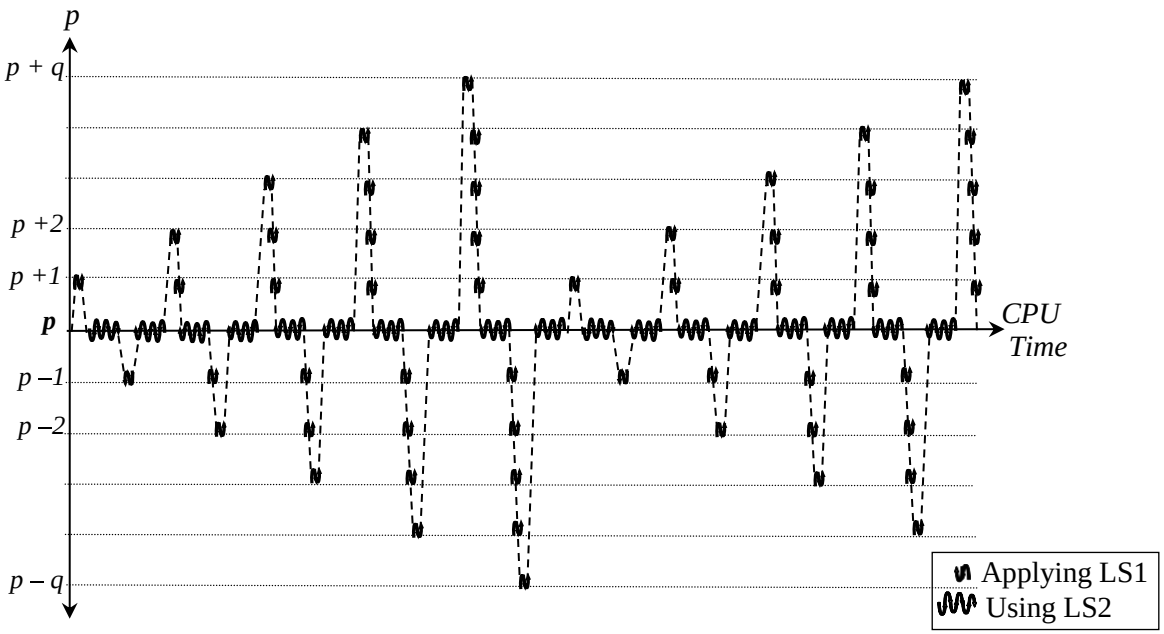


Figure 4.9: The strong perturbation STRONGPERT-V2 (*q is dynamic value*)

The following steps of Enh 2 are as those in Figure 4.3 except that in *Step* 0 we use $q = 1$ and *Step* 2*a*, *Step* 2c and *Step* 3c are replaced by:

Step 2a: Perturb the best solution (X_{best}) by adding randomly one facility in the continuous space encompassed by the covering circle CC_q then set $\overline{p} = \overline{p} + 1$.

Step 2c: If $\bar{p} < p+q$, go to *Step* 2*a* (*it is unchanged*), else apply LS1 and go to *Step* 3.

Step 3c:

If $\bar{p} > p-q$, go to *Step* 3*a* (*it is unchanged*),

else

if $q < q_{\max}$, set $q = q+1$, else set q =1;

go to *Step* 4.

4.5.3 Computational Results of Enh 1 and Enh 2

The same experiment as the one used in the previous section was conducted here. The approaches were also run for 10 times, starting from a random initial solution. The same stopping condition that was used in previous sections was also adopted here. The results were given in Table 4.3. It can be noted that the performance of Enh 1 was slightly better than Enh 2 with respect to the average deviation of the best results from the optimal solutions (i.e., 0.58% vs 0.59%). However, in terms of the average results, Enh 2 outperformed Enh 1 where the average corresponding deviations were 1.30% and 1.59% respectively. The standard deviations of 2.17 and 1.60 for Enh 1 and Enh 2 respectively confirm that Enh 2 was relatively more stable. It is worth noting that this enhancement has made GRADPERT and STRONGPERT-V2 more effective, as the overall deviations were 0.85% and 0.70% as shown earlier in Table 4.1 and Table 4.2 respectively.

Table 4.3: Deviation (%) of the average and the best result from the optimal solution for Enh 1 and Enh 2

n = 439 TSP-Lib p	The optimal solutions (Z)	Enhancement on GRADPERT (Enh 1)				Enhancement on STRONGPERT-V2 (Enh 2)			
		Deviation Average Results	Deviation Best Results	ST DEV	CPU Time*	Deviation Average Results	Deviation Best Results	ST DEV	CPU Time*
10	1716.5099	0	0	0	7.52	0	0	0	5.62
20	1029.7148	0	0	0	10.39	0	0	0	9.59
30	739.19297	0	0	0	16.27	0	0	0	15.43
40	580.00539	0	0	0	50	0	0	0	29.44
50	468.54162	2.62	0.89	6.85	87.89	0.80	0	3.17	100.96
60	400.19527	3.09	0.35	5.09	128.24	2.11	0.35	3.24	81.74
70	357.94553	1.27	1.27	0	115.12	1.13	0.00	1.42	103.25
80	312.5	4.49	1.20	5.72	125.73	4.39	1.98	4.76	109.41
90	280.90256	1.05	0.40	2.46	283.86	0.97	0.40	2.58	106.22
100	256.68019	3.37	1.69	1.62	126.28	3.58	3.19	0.80	146.87
Average	614.21882	1.59	**0.58**	2.17	95.13	**1.30**	0.59	**1.60**	70.85

*: *CPU time when the best solution was found.*

4.6 Incorporating learning within the perturbation-based heuristic

In this section we incorporate learning into our perturbation-based heuristics (Enh 1 and Enh 1). The aim is to identify the most promising values of q and q_{max}. It is worth noting that STRONGPERT (Enh 2) has more flexibility than GRADPERT (Enh 1), as the size of its covering circle is dynamic. Namely, in Enh 2, we can also record information about the radius of the covering circle (the destination circle that we insert the added facilities in).

The learning process consists of two phases as in chapter 3. In the first phase, the information that is mentioned above is recorded during a certain time period (say for instance 25% of the total time) which we call the learning phase. In the second phase, we use the obtained information about q, q_{max} and the level of the covering circle to guide the search during the remaining time of the perturbation-based heuristic, see Figure 4.10 for an illustration.

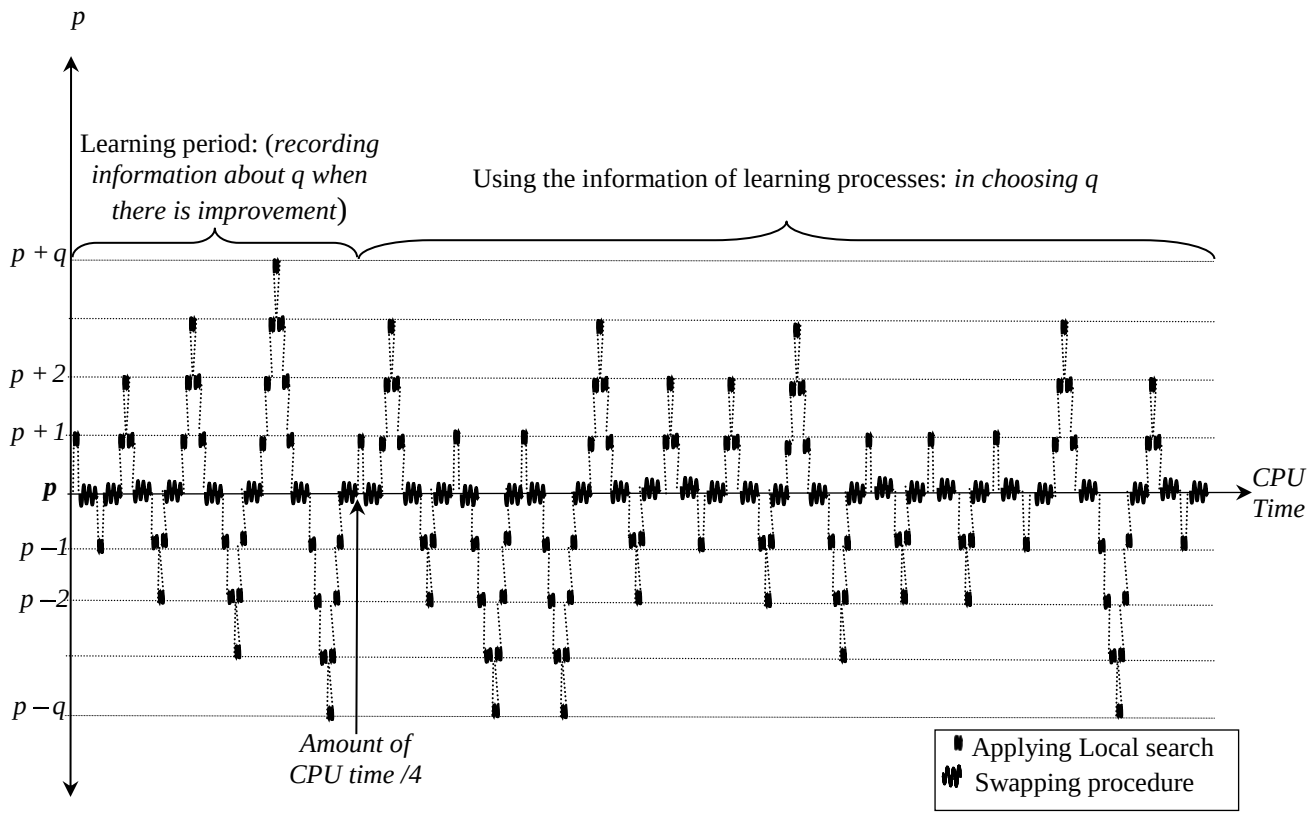


Figure 4.10: The gradual perturbation (*selection of q using the frequency occurrence*)

4.6.1 Phase I: Learning process

In this phase, we gather the information mentioned above.

Determination of the value of (q)

Here, we record the number of times the solution is improved for each value of q (number of added/removed facilities). We also identify the minimum and the maximum q values where the latter relates to $q_{\max}$.

Size of the neighbourhood (the levels of the covering circle)

In STRONGPERT, the destination region (covering circle) that we insert the facilities in is a circle with a dynamically increasing level (radius). The levels used of the covering circle are recorded, whenever the solution improves. In other words, if there is an improvement at a given level, the frequency of using such a level will be increased by one as performed in section 3.5 of chapter 3.

4.6.2 Phase II: Integrating the information within the search

The information (q value) that is recorded in the first phase is then used to guide the search in the perturbation-based heuristic by using the following frequency of occurrence based-scheme as this proved in chapter 3 to be superior than the other scheme such as the range-based scheme.

The information that is recorded in the first phase is then used to guide the search by computing the probabilities of occurrence of each value of q, say $P(q)$ which is based on the number of times a solution is improved. In other words, the higher the probability of a given value of q is, the higher the chance that such a value will be chosen. These probabilities are then used to choose the values of q for both enhancements (Enh 1 and Enh 2) and to choose the level of the covering circle where the new open facilities are inserted in Enh 2.
The same mechanism of using the inverse method, as described in chapter 3, is also implemented here.

4.7 Computational results

In this section, the perturbation-based heuristics namely GRADPERT (Enh 1) and STRONGPERT-V2 (Enh 2) with and without learning were evaluated using the existing data sets (n=439, 575, 783, 1002 and 1323 TSP-Lib) with various values of p (p=10 to 100 with an increment of 10). The optimal solutions for the small data set (n=439) that were provided by Chen and Chen (2009) were used to assess the performance of our methods. For the larger data sets (n=575, 783, 1002 and 1323) no optimal solutions are available, therefore, we assessed the performance of these methods by comparing their results to the best solution of the proposed enhancements. The same stopping criterion used in the previous chapter was also used here. This was the corresponding CPU time for the multi-start procedure using 10,000 iterations. For simplicity, the same multi-start algorithm but with 100 runs was also used as the initial solution in our perturbation-based heuristics.

4.7.1 Comparisons against optimal results (*small data set*)

Table 4.4 shows the optimal solutions for the TSP data set (when n=439), the deviations for both GRADPERT (Enh 1) and STRONGPERT (Enh 2) with and without learning and the multi-start algorithm with 10,000 iterations. The deviations of these methods were computed from the optimal solutions.

Deviation (%) $= \dfrac{(Z_H - Z^*)}{Z^*}.100$ where Z^* refers to the optimal solution and Z_H is the solution found by heuristic 'H'. The experiments show that the performances of Enh 1 and Enh 2 were always better than the one of the multi-start algorithm. In both cases (with and without learning), it can also be observed that the performance of Enh 2 was slightly better than its counterpart Enh1. The average deviation values without learning were 0.59% and 0.65% respectively and 0.32% and 0.47% with learning. However, Enh 1 found 10 optimal solutions out of 20 (i.e., 5 times with learning when $p \leq 40$ and p = 90 and 5 times without learning when $p \leq 40$ and p = 70), while Enh 2 found 9 optimal solutions and the multi-start procedure was unable to find any. It is worth noting that Enh 1 did not always yield better results than its counterpart Enh 2 as shown by the case when p = 100.

In general, in both perturbation-based heuristics (GRADPERT and STRONGPERT) the incorporation of learning has made the search, shown by Enh 1 and Enh 2, to be more

effective than without learning. The average deviation values for Enh 1 with and without learning were 0.47% and 0.65% respectively. For Enh 2, those were 0.32% and 0.59%. In addition, Enh 2 with learning found 5 optimal solutions out of 10 while 4 optimal solutions were found instead when no learning was incorporated into the search.

Table 4.4: Deviation (%) from the optimal solution of GRADPERT, STRONGPERT (with and without learning)

n	*p*	The optimal solutions	Multi-Start for 10,000 iterations	GRADPERT (Enh1)		STRONGPERT (Enh2)	
				No Learning	With Learning	No Learning	With Learning
		Z	Deviation	Deviation	Deviation	Deviation	Deviation
439	10	1716.5099	2.02	**0**	**0**	**0**	**0**
	20	1029.7148	11.42	**0**	**0**	**0**	**0**
	30	739.19297	31.90	**0**	**0**	**0**	**0**
	40	580.00539	18.06	**0**	**0**	**0**	**0**
	50	468.54162	29.41	0.85	0.85	**0.67**	1.18
	60	400.19527	42.45	**0.35**	**0.35**	**0.35**	0.35
	70	357.94553	59.27	**0**	1.23	1.23	**0**
	80	312.5	30.70	3.23	**0.96**	1.98	0.96
	90	280.90256	25.93	0.40	**0**	0.40	0.40
	100	256.68019	27.46	1.69	1.35	1.30	0.35
	Average	614.21882	27.86	0.65	0.47	0.59	**0.32**
	# Optimal		0	5	5	4	5

Bold: *The best solutions found.*

4.7.2 Results on larger data sets (*no known optimal results*)

Four larger existing TSP data sets (*n*= 575, 783, 1002 and 1323 TSP-Lib) were used to assess the performance of our enhancements, see Table 4.5. In these data set, there are no known optimal solutions, therefore, we computed the deviation from the best solution as: Deviation (%) = $\frac{(Z_H - Z_{best})}{Z_{best}}.100$ with Z_H denoting the Z value found by heuristic *'H'* and Z_{best} refers to the best value of Z found by any of the heuristics.

Table 4.5 provided the same information as in the previous table. In general, in both cases (with and without learning), the performance of both perturbation heuristics were relatively much better than the multi-start procedure, as the overall average deviation values confirm that the performances of both perturbation heuristics were always equal or better than the performance of the multi-start algorithm. When learning was not considered, Enh 1

outperformed Enh 2, as the overall average deviations were found to be 0.69% and 0.90% respectively.

Table 4.5: Deviation (%) of Multi-Start, Enh 1 and Enh 2 (with and without learning) from the best solution

n	*p*	Overall best solutions Z	Multi-Start (10,000 Runs)	The multi-start algorithm with 100 runs			
				The Gradual perturbation (Enh1)		The Strong perturbation (Enh2)	
				No Learning	With Learning	No Learning	With Learning
575	10	68.604	0.90	**0**	**0**	**0**	0.90
	20	45.622	3.10	**0**	**0**	0.88	**0**
	30	35.795	8.32	0.85	**0**	0.86	1.05
	40	30.414	13.95	1.95	0.24	0.04	**0**
	50	26.669	15.65	1.07	**0**	0.49	0.39
	60	23.436	19.65	0.54	0.36	2.10	**0**
	70	21.219	13.89	1.02	0.66	1.02	**0**
	80	19.558	24.96	0.24	**0**	2.79	2.26
	90	18.028	22.85	0.84	**0**	0.23	1.45
	100	16.771	27.37	0.59	**0**	0.22	1.41
	Average	30.612	15.06	0.71	0.13	0.86	0.75
783	10	79.313	**0**	**0**	**0**	**0**	**0**
	20	54.002	1.68	1.79	1.58	**0**	1.11
	30	42.974	10.33	0.63	4.19	2.98	**0**
	40	36.321	9.66	0.59	**0**	0.42	0.66
	50	31.357	15.36	**0**	3.34	2.10	**0**
	60	28.128	17.87	0.87	2.60	**0**	0.07
	70	25.446	20.89	**0**	1.46	1.67	2.16
	80	23.665	22.13	0.65	1.97	0.80	**0**
	90	21.759	24.43	**0**	2.69	0.90	2.70
	100	20.430	25.42	0.61	0.61	**0**	1.86
	Average	36.340	14.78	0.51	1.84	0.89	0.86
1002	10	2389.360	0.89	**0**	**0**	**0**	**0**
	20	1607.530	4.79	0.13	0.13	0.13	**0**
	30	1231.360	8.42	0.49	**0**	1.49	**0**
	40	1021.410	18.10	2.24	**0**	2.24	**0**
	50	903.120	16.79	1.63	**0**	1.57	0.03
	60	795.709	22.01	2.57	**0**	0.73	0.59
	70	725.431	17.98	1.83	0.14	0.24	**0**
	80	660.019	22.91	0.14	**0**	1.99	0.14
	90	608.999	27.32	**0**	**0**	**0**	0.22
	100	564.795	28.10	**0**	1.03	1.90	0.21
	Average	1050.773	16.73	0.90	0.13	1.03	0.12
1323	10	2899.420	0.26	0.08	**0**	0.17	**0**
	20	1868.920	5.41	**0**	0.96	0.96	1.15
	30	1477.590	7.51	**0**	1.01	1.46	0.88
	40	1245.410	11.59	1.40	**0**	**0**	0.31
	50	1068.040	15.20	**0**	**0**	0.84	**0**
	60	940.691	12.80	0.10	0.10	0.10	**0**
	70	856	17.84	0.13	**0**	2.22	0.73
	80	783.228	14.09	0.63	**0**	**0**	0.72
	90	719.580	24.18	1.63	2.27	0.38	**0**
	100	663.035	28.61	2.19	0.77	1.98	**0**
	Average	1252.191	13.75	0.62	0.51	0.81	0.38
Overall Average		592.479	15.08	0.69	0.65	0.90	**0.53**
# Best			1	12	**21**	9	18

Bold: *The best solutions found.*

Furthermore, Enh 1 achieved the best solution 12 times whereas its counterpart Enh 2 achieved it 9 times out of 40 times. For the case of learning, the overall average deviation values confirm that the performance of Enh 2 with learning yielded relatively better results than those of Enh 1 where a deviation of 0.53% and 0.65% were found respectively. However, the latter obtained more best solutions than Enh 2 with 21 best solutions were found by Enh 1 against 18 out of the 40 instances respectively.

In both Enh 1 and Enh 2, our experiments confirm that the incorporation of learning into the search has made these enhancements more effective. For GRADPERT, we can see a slight improvement in the overall average deviation where the overall average deviation decreased from 0.69 % to 0.65%. However, it can be observed that the incorporation of learning into STRONGPERT has made Enh 2 with learning much better than Enh 2 without learning, where the overall average deviation decreased from 0.90% to 0.53%, making it the best performer. However, Enh 2 did not always yield better results than its counterpart Enh1 where the latter achieved the best solutions 33 times in total (12 times without learning and 21 with learning out of 40) whereas Enh 2 achieved it 27 times in total (9 and 18 times).

4.7.3 Time performance

This section presents a comparison between the CPU time recorded for Chen and Chen's results (when it is available), the average total CPU time of the multi-start algorithm with 10,000 runs and the average CPU time when the best continuous solution was found say T_H of our perturbation-based heuristics using the above two strategies. The information about when the best solution was usually obtained was recorded for information only as these can be used to design more efficient stopping rules if necessary.

The Deviation is computed from the CPU time of the multi-start algorithm with 10,000 runs as Deviation (%) $= \frac{(T_H - T_{M.S})}{T_{M.S}}.100$, where $T_{M.S}$ refers to the CPU time of the multi-start algorithm and T_H is the time for heuristic H when the best solution is found.

The machine that we used was different to the one used by Chen and Chen. For a fair comparison between our enhancements and their results, the transformation that was given by Dongarra (2013) was used. This was conducted in the same way as explained in chapter 3.

In general, from Table 4.6, we can conclude that the overall deviations of CPU time for the perturbation heuristics when the best solution was found increased with *n*. For instance, in the case of Enh 1 without learning, the overall deviations were -73.83% and -46.99% when *n* = 439 and 1323 respectively.

In brief, the overall average deviation values of CPU time for both perturbation heuristics when the best solution was found confirm that the best solution of the perturbation-based enhancements (Enh 1 and Enh 2) required a CPU time which was less than 50% of the multi-start algorithm CPU time when the best was found. In other words, 50% of the CPU time of the multi-start algorithm could be large enough to be used as a stopping condition for our perturbation-based enhancements. Besides, this also showed that using another stopping rule such as the number of successive cycles without improvement could be one way forward. This can be useful as it is problem dependant instead of being fixed from the outset.

Table 4.6: Average CPU time of the Multi-Start algorithm (for *p*=10 to 100 in increment of 10), Deviation (%) of CPU time for Enh1 and Enh2 (with and without learning)

n	Average total CPU time (10,000 iterations) (secs)	Deviation (%)					
		Perturbation-Based Enhancement (*initial solution based on 100 restarts*)				Chen and Chen's results (Continuous Solutions)	
		Enh 1*		Enh 2*		Improved relaxation (*k*=7)	Binary relaxation (*k*=6)
		No Learning	With Learning	No Learning	With Learning		
439	1497.56	-73.83	-67.13	-87.22	-67.19	-88.37	-98.72
575	1681.81	-59.99	-47.79	-60.68	-36.43	N/A	N/A
783	2762.45	-52.33	-54.76	-56.36	-46.32	N/A	N/A
1002	4398.09	-69.23	-65.43	-78.29	-58.59	N/A	N/A
1323	5662.98	-46.99	-67.06	-77.49	-59.47	N/A	N/A
Average	3200.58	-60.47	-60.43	-72.01	-53.60	N/A	N/A

*: *CPU time when the best solution was found.*
k:is the best recorded value in Chen and Chen (2009).

4.8 Comparison between VNS and perturbation-based heuristic

In this section, the overall best solutions of the VNS-based heuristic approach of chapter 3 (VNS (*CNV3*) and VNS2 (*FNV4*) with and without learning) and the perturbation-based heuristic (GRADPERT (Enh 1) and STRONGPERT-V2 (Enh 2) with and without learning) were compared just for information.

Table 4.7 shows the deviation of these heuristics from the best results. It can be observed that the performance of VNS was slightly better than the perturbation heuristic where the overall average deviations were 0.16% and 0.18% respectively. In addition, VNS found 35 best solutions whereas the other found 30 out of 50. However, it is worth noting that the VNS-based heuristic did not always yield better results than its counterpart as shown when n = 575 and 1002. Note that the overall best solution was chosen from the best of any of the heuristics used in this work for comparison purposes. These results also highlighted that it could be worth exploring further the idea of perturbation.

Table 4.7: Deviation (%) of the best solution of VNS and the perturbation-based heuristic from the best solution

n	*p*	Overall best solutions	VNS %	Perturbation %
439	10	1716.510	**0**	**0**
	20	1029.710	**0**	**0**
	30	739.193	**0**	**0**
	40	580.005	**0**	**0**
	50	471.699	**0**	0.18
	60	401.591	**0**	**0**
	70	357.946	**0**	**0**
	80	312.552	**0**	0.94
	90	280.903	0.40	**0**
	100	257.570	0.05	**0**
	Average	614.768	**0.05**	0.11
575	10	67.926	**0**	**0**
	20	45.622	**0**	**0**
	30	35.556	**0**	**0**
	40	30.265	**0**	0.49
	50	26.173	**0**	0.56
	60	23.436	0.79	**0**
	70	21.059	**0**	0.76
	80	19.266	1.52	**0**
	90	17.805	0.67	**0**
	100	16.621	**0**	0.54
	Average	30.373	0.30	**0.24**
783	10	79.313	**0**	**0**
	20	53.461	**0**	0.43
	30	42.395	**0**	0.96
	40	35.962	**0**	1
	50	31.357	0.17	**0**
	60	28.053	**0**	0.27
	70	25.446	**0**	**0**
	80	23.560	**0**	0.44
	90	21.710	**0**	0.23
	100	20.334	**0**	**0**
	Average	36.159	**0.02**	0.33
1002	10	2389.360	**0**	**0**
	20	1607.530	0.13	**0**
	30	1231.360	**0**	**0**
	40	1021.410	0.88	**0**
	50	901.455	0.53	**0**
	60	795.709	0.73	**0**
	70	725.431	0.24	**0**
	80	660.019	0.72	**0**
	90	604.494	**0**	**0**
	100	559.017	**0**	0.01
	Average	1049.579	0.32	**0.00**
1323	10	2897.490	**0**	0.07
	20	1868.920	0.96	**0**
	30	1466.970	**0**	0.72
	40	1236.380	**0**	0.34
	50	1060.820	**0**	0.08
	60	940.691	0.13	**0**
	70	844.967	**0**	0.57
	80	774.764	**0**	0.21
	90	719.580	0.15	**0**
	100	662.936	**0**	0.02
	Average	1247.352	**0.12**	0.20
Overall Average		595.646	**0.16**	0.18
# Best			35	30

4.9 Summary

In this chapter, a brief review of the perturbation-based heuristic was first given followed by the three moves that were used in this meta-heuristic. The idea behind the perturbation is to allow the number of facilities to be higher and lower than p in order to act as a filtering process where the best facilities have the tendency to remain in the promising set. Two types of perturbations (GRADPERT and STRONGPERT) were designed followed by an enhancement which included the use of dynamic values of q leading to Enh 1 and Enh 2. Furthermore, the use of learning was incorporated within the search to make the search more adaptive.

In the computational results section, five TSP data sets with n = 439, 575, 783, 1002, 1323 and p varying from p =10 to 100 with a step of 10 were used as a platform to test our proposed perturbation (Enh 1 and Enh 2). The obtained results of the small data set (n=439) showed that Enh 2 with and without learning outperformed its counterpart Enh 1, where the average deviation were 0.32% and 0.59% vs 0.47% and 0.65% respectively. However, for the large data set (n=575, 783, 1002 and 1323), the performance of Enh 1 was better than Enh 2, but when learning is incorporated, the overall average deviation of Enh 2 has clearly improved to outperform Enh 1, where its deviation decreased from 0.90% to 0.53% vs 0.69% and 0.65% for Enh 1. Finally, the comparison between VNS and perturbation-based heuristic was also presented, where the performance of the VNS-based heuristic was found to be slightly better than the perturbation-based heuristic.

The next chapter will deal with a newly developed local search that is originally developed for the multi-source Weber problem.

Chapter 5

Reformulation Local Search-Based Approaches

5.1 Introduction

It should be noted that the idea of using an optimal discrete solution as an initial solution for the continuous case is explored. This is extended to cover the new concept of reformulation local search (RLS) originally applied to the multi-source Weber problem, which is then adapted to solve this related continuous location problem. This is followed by systematically generating a tighter *Z* value as the new upper bound when solving the discrete problem. Some forms of relaxation in our stopping criteria are also analysed. Two enhancements on the RLS procedure are designed using forbidden regions followed by extensions that incorporate the idea of injection points for diversification purposes and memory management for controlling the size of the augmented discrete problem. Computational experiments are conducted yielding interesting results. Finally, the solutions of the best enhancements for VNS and the perturbation-based heuristics are used as starting solutions for the best RLS with the aim to produce good quality solutions for benchmarking.

5.2 The discrete-based approach

The idea of using the optimal or heuristic solution of the discrete problem as a starting solution for the problem in the continuous space has been considered by Hansen *et al.* (1998) and also Gamal and Salhi (2001) for the multi-source Weber problem. The earlier paper used an optimal method for the discrete problem whereas the latter adopted a powerful heuristic instead. In this section we adapt such a methodology focusing on the former approach to solve the continuous *p*-centre problem.

It is worth noting that the optimal solution of the discrete case is obviously greater than or equal to the one found on the continuous space. This statement follows from the basic optimisation theory where $\underset{X \in S_C}{Min}\ f(X) \leq \underset{X \in S_D \subset S_C}{Min}\ f(X)$. In our case, S_D refers to the discrete space contained which is contained into the continuous space S_C and $f(X)$ is the maximum

radius using the facilities located at $X = (X_1,..., X_p)$. As an illustration see Figure 5.1, where the optimal solution (R_{max}) of the discrete case and the solution ($\bar{R}_{max}$) of continuous case, which is found after applying the local search on the optimal discrete solutions, are shown in Figure 5.1 (*a*) and Figure 5.1 (*b*) respectively with $\bar{R}_{max} < R_{max}$. It is worth noting that in the continuous solution each circle can be determined by three demand points (critical points) on its circumference (vertices of an acute triangle) or by two critical points on the two ends of a diameter. However, in the discrete solution the circle can be defined by the location of two demand points, one as its centre and the other as a critical point on its circumference. Note that in special cases there may be several points which happen to lie on the circumference as well.

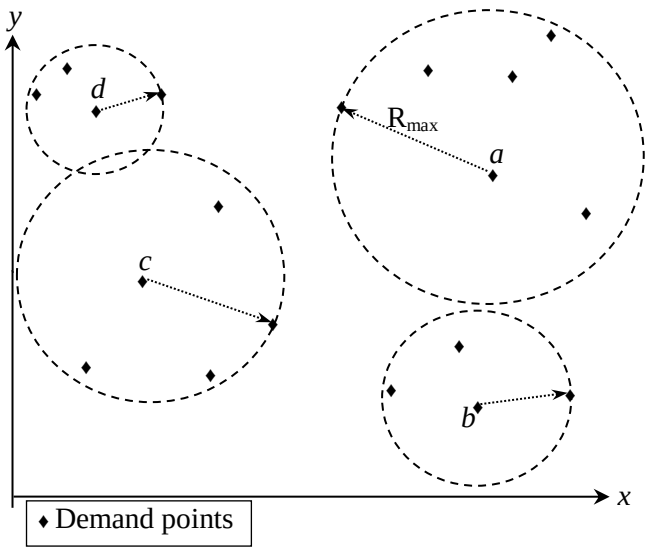


Figure 5.1 (*a*): An optimal solution of a *4*-centre discrete location problem

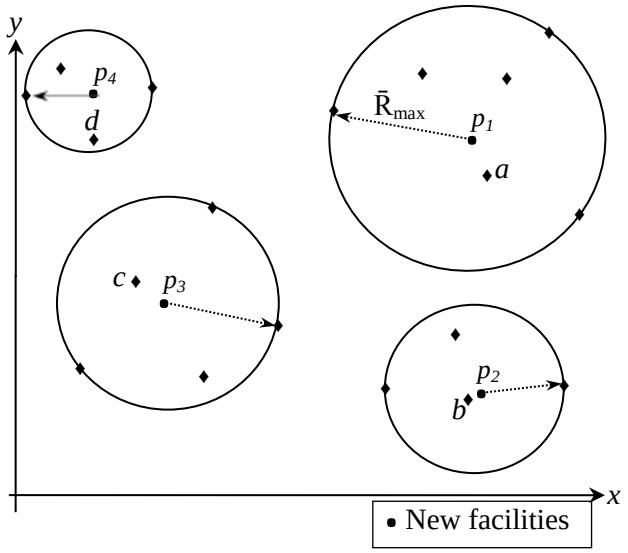


Figure 5.1 (*b*): A feasible continuous solution of the same *4*-centre location problem ($\bar{R}_{max} < R_{max}$)

There are two general approaches for solving optimally the vertex *p*-centre problem, as reviewed in chapter 1. These include: (i) the classical ILP-based-approach and (ii) the set covering problem-based (SCP) approach. The latter is known to be relatively more efficient, and hence, is the one we shall pursue here. It is based on the principles of Minieka's algorithm (1970) whose idea is to search over a range of coverage distances in order to find the smallest value that covers all the demand nodes.

Daskin (1995) adopted this approach on a general graph using a binary search that requires an initial lower bound (L) and an upper bound (U). In this algorithm all distances are integer. In the original implementation $L = 0$ and $U = \underset{i,j}{Max}\, d(P_i, P_j)$. A bisection method is then adopted where the coverage distance $D = \frac{L+U}{2}$ is used to solve the Set Covering Problem (SCP), which is described in Figure 5.2 yielding the optimal number of facilities *r*. If $r \le p$ (i.e., the solution is feasible for the p-centre problem), set $U = D$; else (i.e., $r > p$ and the solution is infeasible) set $L = D + 1$. This procedure is repeated until $U = L + 1$ and the optimal solution U is found. Al-Khedhairi and Salhi (2005) proposed, among other enhancements, tighter initial bounds for L and U using $L = \underset{j}{Max}\ \underset{i}{Min}\ d(P_i, P_j)$ and $U = \underset{i}{Min}\ \underset{j}{Max}\ d(P_i, P_j)$. The authors also proposed a more efficient implementation using $D = G(\frac{L+U}{2})$ where $G(x)$ represents the nearest element to *x* in the distance matrix. In addition, when the set $S = \{d(P_i, P_j) : L < d(P_i, P_j) < U\} = \varnothing$ (i.e., there are no distance values between U and L) the optimal solution is then set to U and the search stops even if $U - L > 1$. Salhi and Al-Khedhairi (2010) improved this implementation further by using $U = Z_H$ as the solution of a powerful meta-heuristic namely the multi-level heuristic originally designed by Salhi and Sari (1997) used for a class of routing problems. The lower bound L is derived using $L = \alpha U$ with $\alpha = 0.7$ or 0.8. The tighter Z_H is, the closer α is to 1. The strength of this scheme is that even if αU does not yield a lower bound, it will be systematically an upper bound instead. In this case, U takes such a value and the lower bound is recomputed again using $L = \alpha U$ again until a proper range (L, U) is identified for the bisection procedure to proceed. In other words, there are no redundant computations in the updating of L. The main steps of this enhanced SCP-based algorithm for the vertex *p*-centre problem are given in Figure 5.2.

Step 1: Choose α and set $U = Min(\underset{i}{Min}\,\underset{j}{Max}\, d(P_i, P_j), Z_H)$ and $L = Max(\underset{j}{Max}\,\underset{i}{Min}\, d(P_i, P_j), \alpha Z_H)$
Construct $S_0 = \{d(P_i, P_j) : L < d(P_i, P_j) < U\}$ and set $S = S_0$.

Step 2: Calculate $D = G(\frac{L+U}{2})$ where $G(x)$ represents the element in S nearest to x.

Step 3: Solve the set covering problem (SCP) for the covering distance D, and let r be the number of facilities found.

a) If the solution is feasible for the p-centre problem (i.e. $r \leq p$), set $U = D$,
b) Else (i.e., $r > p$) set $L = D$.

Step 4: Set $S_0 = \{d(P_i, P_j) \in S : L < d(P_i, P_j) < U\}$ and $S = S_0$.
If S= ∅, the optimal solution is U and stop; else go to *Step* 2.

Figure 5.2: The enhanced SCP-based algorithm for the vertex *p*-centre problem

Further details on the vertex *p*-centre location problem can be found in Mladenović *et al.* (2003) and Salhi and Al-Khedhairi (2010) and references therein.

This discrete-based approach, which we refer to as DBA for short, is given in Figure 5.3.

Step 1: Solve the vertex *p*-centre problem using the SCP based-approach as given in Figure 5.2 (or a heuristic if necessary).

Step 2: For each facility and its associated demand points solve the *1*-centre problem using the Elzinga-Hearn algorithm or the enhancements proposed in chapter 2.

Step 3: Apply the 'locate-allocate' procedure with *Step* 2 used to solve the corresponding *1*-centre problems until no further improvement is found.

Figure 5.3: The Discrete Based Approach (DBA)

Computational results

The existing data set (*n*=439, TSP-Lib) with *p* =10 to 100 with an increment of 10 was used to assess the benefits of DBA. The optimal solutions for this particular data set were reported in Chen and Chen (2009) whose algorithms were discussed in chapter 1 subsection

1.5.3. Table 5.1 shows that the average deviation of the optimal discrete solutions from the optimal continuous ones was around 25.22% with the average deviations were in the range the optimal continuous ones (15% − 41%). It can be observed that when we applied the local search after solving the vertex *p*-discrete problem (i.e., DBA) the average deviation decreased to 18.85%, though the range remained large (6%−36%). In terms of computational effort, we also determined the deviation in percentage from the total CPU used to find the optimal discrete solution against the percentage of time for obtaining the continuous solution. In general, a larger proportion of the time was found to be consumed in solving optimally the vertex *p*-centre problem (92%). However, the CPU time for determining the continuous solution increased with *p*. For instance, when $p = 10$ it was around 1% of the total time only, but when $p = 100$, the time for computing the continuous solution (*Steps* 2 and 3 of Figure 5.3) reached around 26%; see Table 5.1.

To illustrate the deviation (%) of the optimal discrete solution and the continuous one found by DBA, a line chart was also displayed in Figure 5.4. It may be noted that in this particular experiment, the objective function values (from p=10 to 100) of the discrete case were found to be always greater than those obtained in the continuous phase of DBA. However, this claim may not always be true as theoretically we can only expect that the solution cannot get worse.

Table 5.1: Deviation (%) of the solutions from the optimal solution and Deviation (%) of the CPU time for the optimal discrete solution and the continuous solution

n =439 TSP-Lib	The optimal solutions	The objective function (Z)			CPU Time				
		The Optimal Discrete	DBA	Improvement (%)	Total of CPU Time	The optimal Discrete (*Step* 1 of DBA)		Continuous phase (*Steps* 2 and 3 of DBA)	
p	Z	Deviation%	Deviation%		(secs)	(secs)	Deviation %	(secs)	Deviation %
10	1716.5099	14.87	6.39	8.48	4.22	4.16	98.65	0.06	1.35
20	1029.7148	15.14	9.42	5.72	5.42	5.29	97.56	0.13	2.44
30	739.19297	19.53	15.25	4.28	5.07	4.96	97.81	0.11	2.19
40	580.00539	15.82	11.73	4.09	4.68	4.37	93.32	0.31	6.68
50	468.54162	20.38	17.02	3.36	4.93	4.83	97.97	0.10	2.03
60	400.19527	24.94	20.99	3.95	5.42	5.17	95.44	0.25	4.56
70	357.94553	32.52	25.96	6.56	6.30	5.76	91.40	0.54	8.60
80	312.5	31.94	21.54	10.40	5.73	5.41	94.45	0.32	5.55
90	280.90256	40.72	35.63	5.09	4.80	3.64	75.74	1.17	24.26
100	256.68019	36.36	24.55	11.80	5.99	4.44	74.08	1.55	25.92
Average	614.21882	25.22	18.85	6.37	5.26	4.80	91.64	0.45	8.36

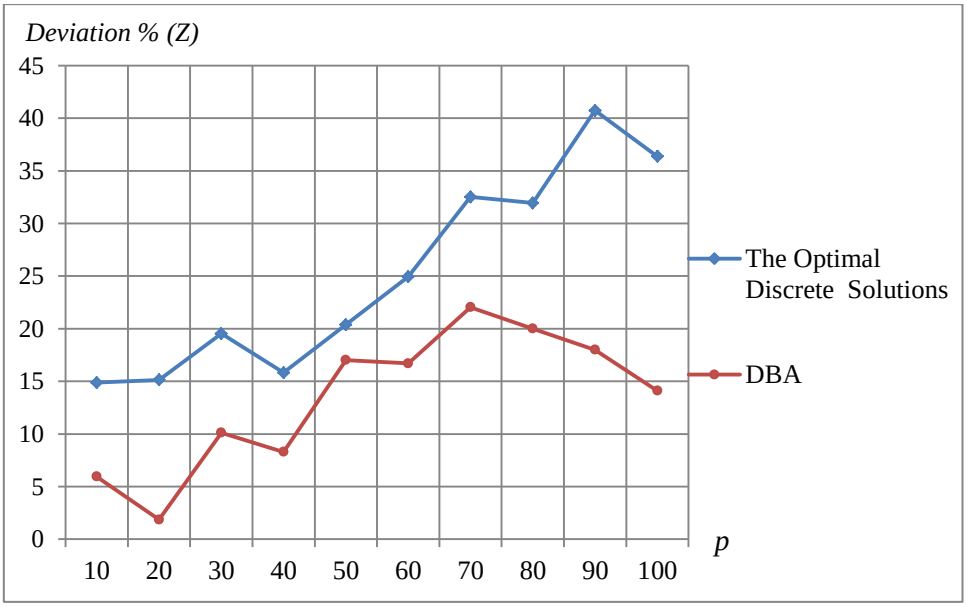


Figure 5.4: Deviation (%) from optimality of the discrete solution and corresponding continuous solution found by DBA (n=439, p = 10 to 100)

5.3 Adaptive RLS-based heuristics

In this study, we first adapt the new reformulation local search (RLS) recently developed by Brimberg *et al*. (2014) for the multi-source Weber problem, to solve the continuous p-centre location problem. This will serve as a basis for our enhanced version which we shall present in the next section. The basic RLS is summarised as follows: We first start from an optimal (or heuristic) solution of the discrete problem followed by a mechanism, such as a local search, to turn such a solution into a continuous one. These new continuous locations are then added as potential sites to make up the augmented discrete location problem. This discrete problem, which still has the same number of demand points, is then solved again optimally (if possible) or heuristically. The new obtained solution of the discrete case is used again as a starting point for the continuous problem. This shift between the discrete and the continuous spaces continues until no improvement is found in solving either the continuous problem or the augmented discrete. The continuous case could be identified by the total cost, the changes in the allocation of the demand points to their regions or simply changes in the facility configuration.

5.3.1 Basic RLS for the continuous *p*-centre problem

In this study, we solve the vertex *p*-centre problem optimally using the efficient implementation of the SCP-based approach, as given in Figure 5.2. Note that in Brimberg *et al*. (2014) a heuristic solution was used at this stage instead as the discrete *p*-median problem is relatively much harder to solve optimally than its counterpart the *p*-centre problem. For the local search in the continuous space, the Cooper's locate-allocate procedure is used in the allocation of the demand points. The enhanced version of the original Elzinga-Hearn algorithm, as proposed in chapter 2 (*V*3 or *V*4), is used to solve the *1*-centre problems. The main steps of the basic RLS for the continuous *p*-centre problem are given in Figure 5.5.

Step 1: Solve optimally the vertex *p*-centre problem with $E = \{P_1, ..., P_n\}$ as the initial set of potential sites. Let X_D be the optimal locations for these *p* facilities and construct the p sets of allocation by assigning each demand point to its nearest facility.

Step 2: Solve the continuous problem

- *Step 2a*: For each of the *p* independent sets of allocations, solve the continuous *1*-centre problem optimally
- *Step 2b*: Allocate each demand point to its nearest facility using the new set of facility locations found in *Step* 2*a*. If there are still changes in the demand point allocation return to *Step* 2*a*, otherwise let X_C be the new facility configuration of the continuous problem and go to *Step* 3.

Step 3: If $X_C = X_D$ then a local minimum is found ($X^* = X_C$) and stop; otherwise set $E = E \bigcup X_C$.

Step 4: Solve optimally the augmented vertex *p*-centre problem using E as the new set of potential sites and let X_D be the optimal locations for these *p* facilities.

If $X_D = X_C$, set $X^* = X_D$, and stop; otherwise construct the p sets of allocations by assigning each demand point to its nearest facility and go to *Step* 2.

Figure 5.5: A basic reformulation local search (RLS) for the continuous *p*-centre problem

5.3.2 Empirical results for RLS and DBA

The heuristics were coded in C++ and run on a PC computer with an Intel Core 2 Duo processor, 2.0 GHz CPU and 4G memory. For the optimal solution of the discrete problem (*Steps* 1 and 4), an integrated C++ code, with CPLEX (version 12.5.1) incorporated within it, was used.

Table 5.2 shows the deviations from the known optimal continuous solution for the DBA and the basic RLS, as well as their corresponding CPU times (in seconds). In general, it can be noted that the deviation values increased with p for both algorithms. From Table 5.2, we can also conclude that RLS has improved the solution of DBA by over 3% on average with a maximum gain of nearly 8% was recorded when p =70.

In terms of CPU time, RLS required nearly double the amount used by DBA, with average values of 4.22 and 7.95 secs respectively. Table 5.2 also shows that the average number of switches, which represents the number of calls to the vertex p-centre problem between the discrete and the continuous cases, was 2.3 though, in some cases, this number could reach up to 4 (eg., case of p =50). In addition, we also recorded the total number of Cplex calls, as some cases were easier to solve than others, where the largest number was reported to be 55 while the average is only 36.

Table 5.2: Deviation (%) of the DBA and RLS from the optimal solution

n = 439 TSP-Lib	The optimal solution	(Z) DBA (1)		Reformulation Local Search (RLS)					Total CPU Time
				Z (2)	Improvement (%)	CPU Time	# Switches	# Cplex Calls	
p	(Z)	Deviation % (1)	CPU Time	Deviation % (2)	(1 - 2)	(sec)			
10	1716.5099	5.95	4.22	5.95	0	3.57	1	26	7.79
20	1029.7148	1.87	5.42	1.76	0.11	5.86	2	38	11.28
30	739.19297	10.11	5.07	10.11	0	3.82	1	22	8.89
40	580.00539	8.30	4.68	7.94	0.35	6.69	2	36	11.37
50	468.54162	17.02	4.93	9.13	7.89	14.56	4	55	19.49
60	400.19527	16.70	5.42	9.15	7.55	10.14	3	46	15.56
70	357.94553	22.04	6.30	14.11	7.93	11.46	3	41	17.77
80	312.5	20	5.73	18.93	1.07	11.07	3	41	16.80
90	280.90256	17.99	4.80	14.84	3.15	7.11	2	29	11.91
100	256.68019	14.11	5.99	12.01	2.10	5.24	2	26	11.23
Average	614.21882	13.41	5.26	10.39	3.02	7.95	2.3	36	13.21

Figure 5.6 showed a summary of the comparison between the DBA and the RLS solutions, in terms of deviation (%) from the optimal continuous solution. In these particular experiments, RLS proved to outperform DBA when $p \geq 50$.

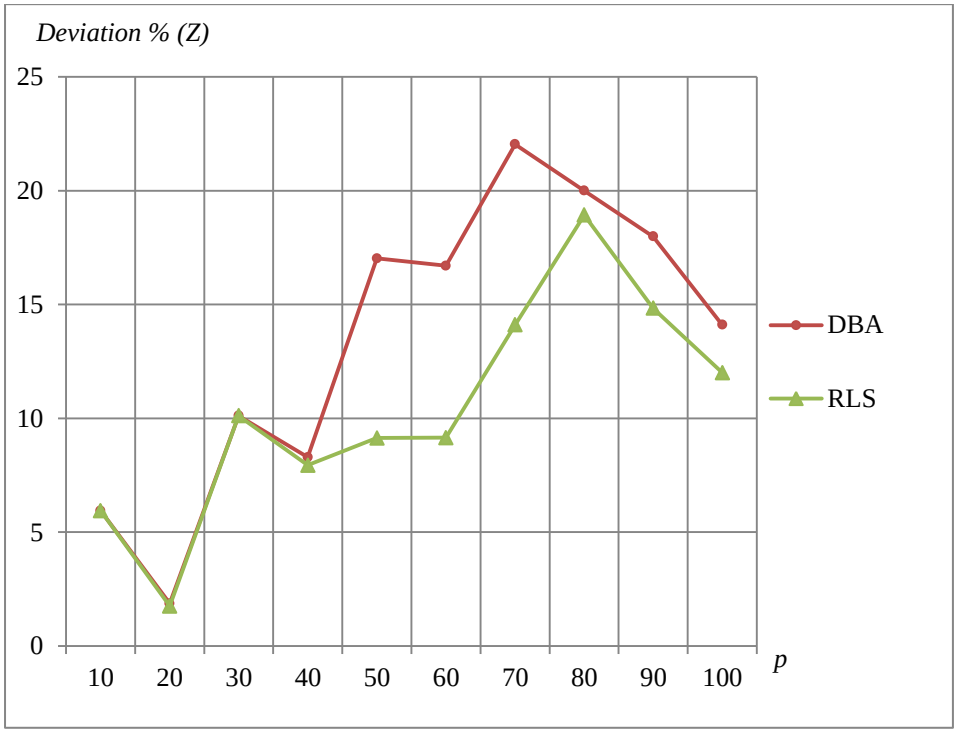


Figure 5.6: Deviation (%) of the solutions from the optimal solution for the DBA and RLS solution

5.3.3 The use of a tighter upper bound (*U*)

In *Step*1 of Figure 5.2, we set Z_H as the best solution of the multi-start algorithm using 100 runs and $\alpha = 0.8$ as suggested by Salhi and Al-Khedhairi (2010).

As RLS usually yields a new solution (Z') in the 'augmented' discrete space which must be less than or equal to the previous Z value (i.e., $Z' \leq Z$), we take this observation into account when updating U. In other words, we set $U = Z'$ at the next round when solving the vertex *p*-centre problem instead of the original value of U. It is worth noting that this updating tends to reduce the number of Cplex calls which often leads to a reduction in the CPU time as shown in Table 5.3. It can be confirmed that the overall average number of Cplex calls when using the new updating scheme of U and L in the SCP-based approach was reduced to around 10% with the largest difference of 8 calls (55 - 47) observed in the case of $p = 50$. In addition, in terms of CPU time, this updating produced an overall average saving of around 15%.

Table 5.3: CPU Time (sec) and the total number of Cplex calls for both cases (with and without updating of U)

n = 439 TSP-Lib	# Switches	Without updating		With updating	
p		# Cplex Calls	CPU Time	# Cplex Calls	CPU Time
10	1	26	7.79	24	6.41
20	2	38	11.28	35	9.65
30	1	22	8.89	22	9.05
40	2	36	11.37	29	9.80
50	4	55	19.49	47	16.52
60	3	46	15.56	42	13.51
70	3	41	17.77	37	13.81
80	3	41	16.80	38	13.94
90	2	29	11.91	26	9.23
100	2	26	11.23	24	10.21
Average	2.3	36	13.21	32.4	11.21

The number of Cplex calls and CPU time (sec) for both cases (with and without updating) were also displayed in Figure 5.7 and Figure 5.8 respectively.

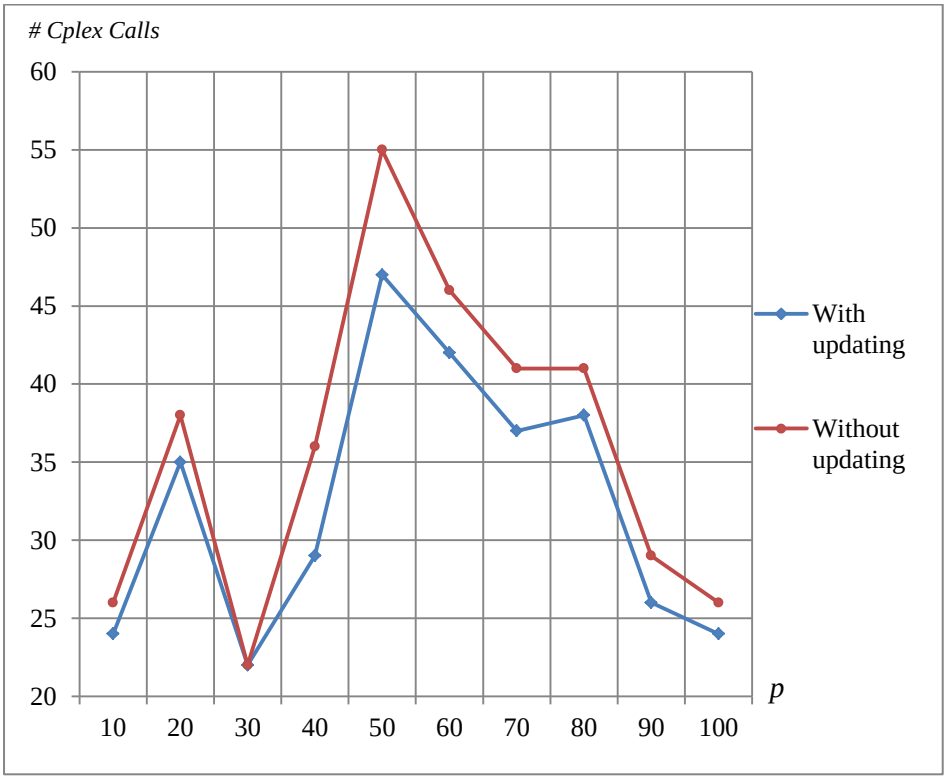


Figure 5.7: Total number of Cplex calls for both cases (with and without updating)

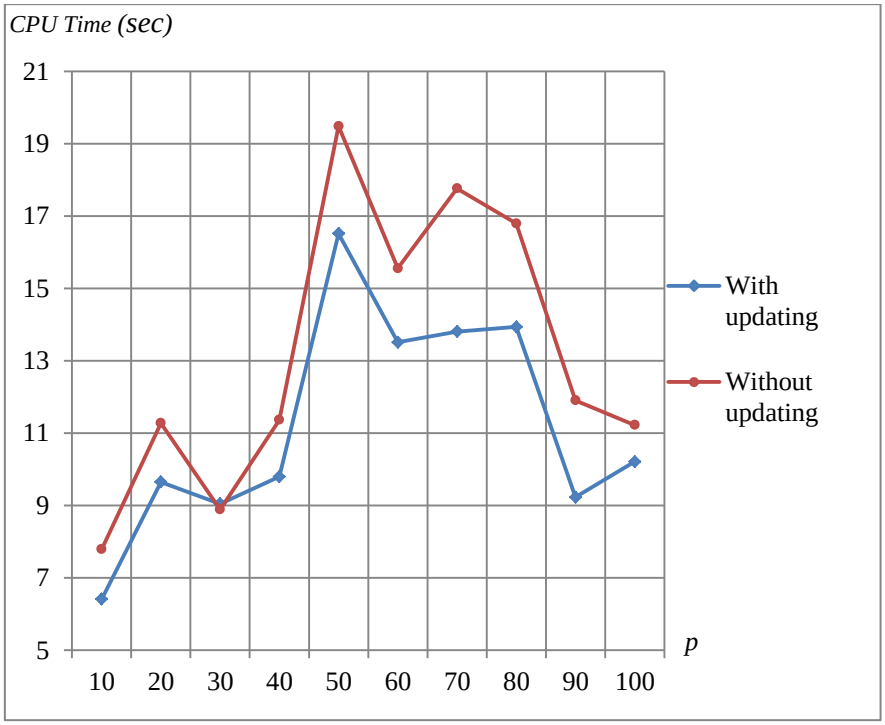


Figure 5.8: Total CPU time for both cases (with and without updating)

To illustrate the effect of the updating scheme on the number of switches, consider for example the case when $p = 50$. Table 5.4 and Figure 5.9 show the number of Cplex calls for each switch with and without the scheme. It can be observed that at the beginning (the first switch) the number of Cplex calls in both cases were equal, then this number using the updating scheme started decreasing when compared to the other one. It is worth noting that the overall benefit of using such a scheme will be improved if the number of switches becomes large.

Table 5.4: Number of Cplex calls for each switch with and without updating (case of p=50)

n = 439 TSP-Lib	# Cplex calls for each switch	
# Switches	Without updating	With updating
0 (starting)	10	10
1	10	10
2	11	8
3	11	9
4	13	10
$\sum$	55	47
Average	11	9.4

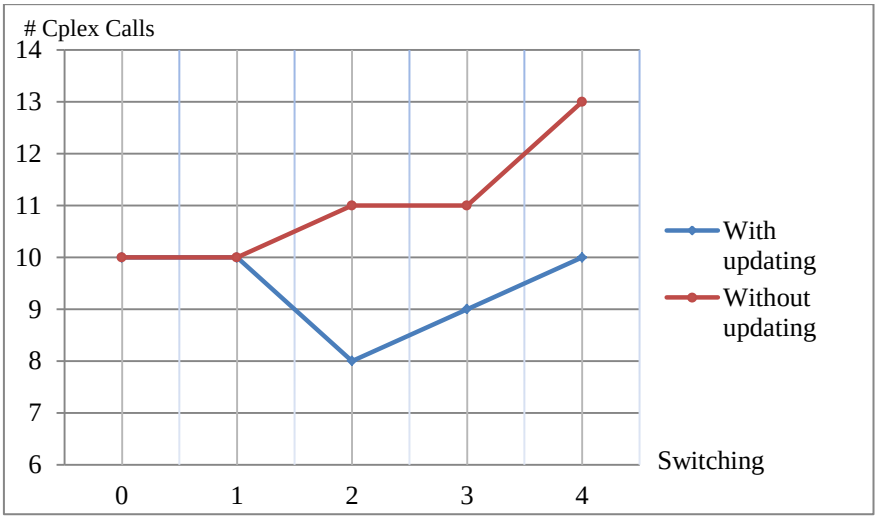


Figure 5.9: Number of Cplex calls for each switch for both cases (with and without updating) when p=50

5.3.4 Introducing flexibility within the stopping criteria

The commonly used stopping criterion is to terminate the search when no improvement is found either in the discrete solution or in the continuous one whichever comes first. Because of the special characteristics of the solution for the p-centre problem where the optimal solution value is defined by one facility location only namely the circle with the largest radius, two other stopping rules are introduced here to respond to this particular characteristic. In the special situation when more than one circle happens to have the same largest radius (case of a tie) these rules would still be valid. These two options are described as follows:

a) Allowing one extra iteration

When there is no improvement in the objective function of the discrete or the continuous space, we allow an extra switch to be performed by adding the new continuous points that are different from the current potential sites. This situation arises when a new facility configuration is found but without an improvement in the Z value. For instance, Figure 5.10 (*a*) shows four locations of the optimal discrete problem namely p_1, b_1, c_1 and d_1. Consider p_1 as the continuous location that has been added to the problem in the previous switch. Figure 5.10 (*b*) also shows four feasible locations in the continuous space (p_1, p_2, p_3 and p_4) for the same problem. It can be seen that the solution cannot be improved as $R_{max1} = \bar{R}_{max1}$.

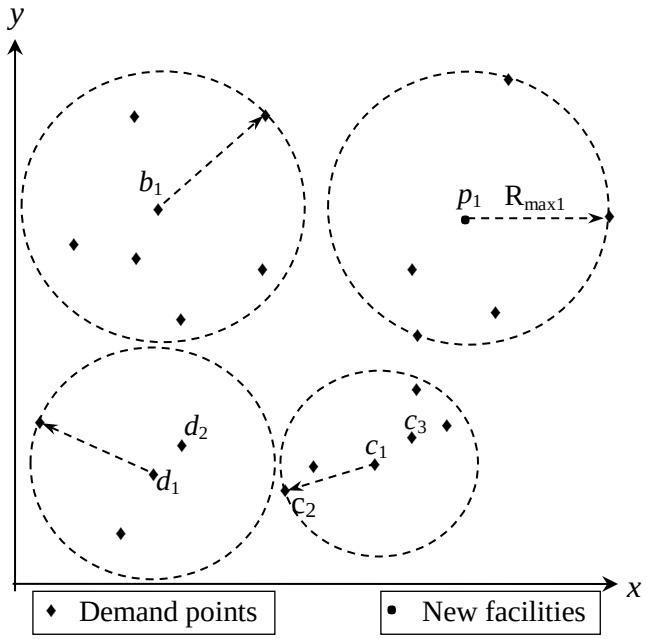


Figure 5.10 (*a*): An optimal solution of the discrete case for a *4*-centre location problem

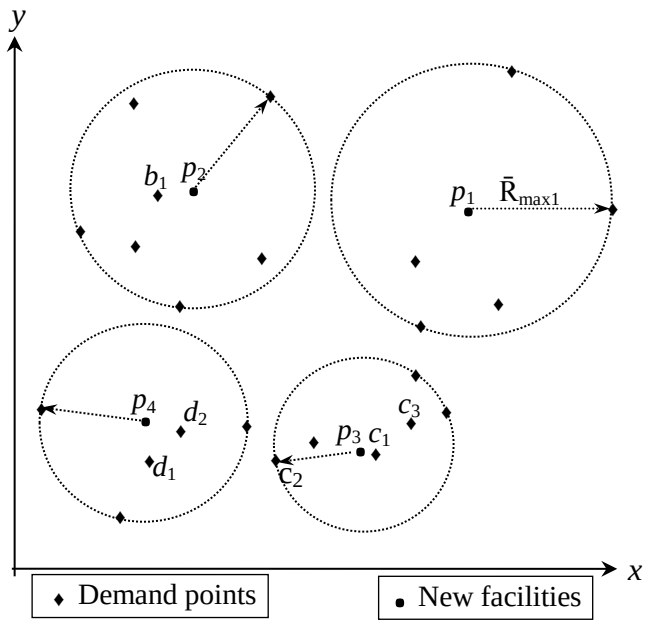


Figure 5.10 (*b*): A feasible solution of the continuous case for a *4*-centre location problem

In Figure 5.11 (*a*), adding the three new continuous locations (p_2, p_3 and p_4) to the problem cannot improve the discrete solution as $R_{max1} = R_{max2}$, but the configuration is different and the discrete solution is also different as shown by (p_1, p_2, c_3 and d_2) instead of (p_1, b_1, c_1 and d_1), see Figures 5.11 (*a*) and 5.10 (*a*) respectively. However, the solution is now improved when the local search is applied on the new discrete solution as shown in Figure 5.11 (*b*) yielding $\bar{R}_{max2} < R_{max2}$.

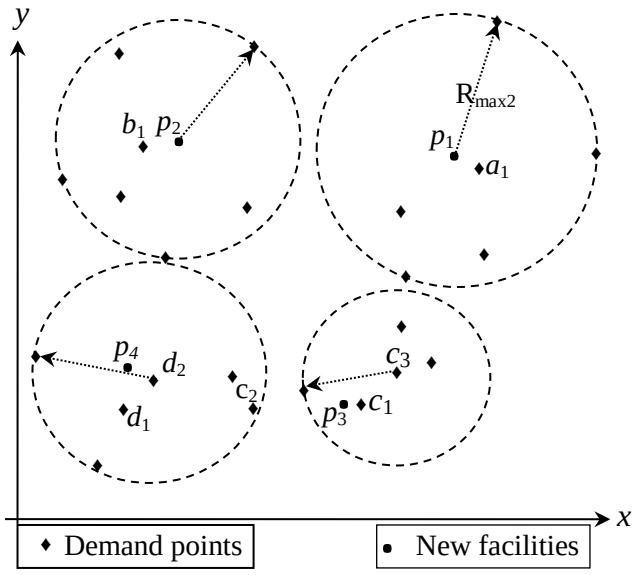


Figure 5.11 (*a*): An optimal solution of the same *4*-centre location problem but with different configuration

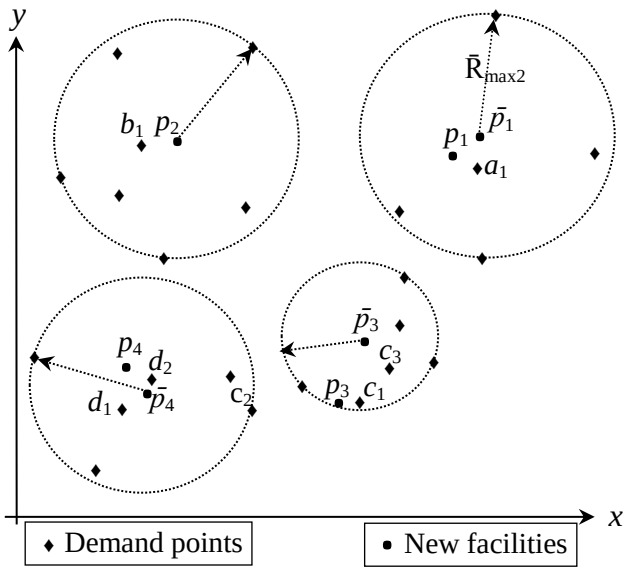


Figure 5.11 (*b*): A better solution of the same *4*-centre location problem

This new stopping criterion is found to allow the search to continue and hence improve the solution in some cases. This happens rarely because when we add a new continuous location to the problem as potential site we might get another optimal solution for the augmented discrete location problem but with a different configuration. In other words, the largest circle could be the same while other circles may be different. When the local search is applied on this new optimal solution, the solution of the continuous problem might improve. This characteristic is unlikely to exist in the multi-source Weber problem but can happen in the continuous *p*-centre problem.

b) No change in the solution configuration at the continuous space

The idea here is that we allow the exchange between the discrete and the continuous cases to continue until no new continuous location is found irrespective even if the Z value remains unchanged. However, using this stopping condition will require the number of switches to be relatively large compared to the previous stopping rules. This additional flexibility could provide more opportunity to improve the solution as will be shown in the following computational results. This rule could be considered to be a generalisation of rule (*a*). Table 5.5 shows the number of the new points that were generated at each switch. For instance, when p=50 the first switch when applying the local search on the optimal discrete solution, produced 47 new continuous locations and, when these points were added to the problem, 10 new other points were then found.

Table 5.5: The number of new continuous locations when allowing the next switch

n = 439 TSP-Lib	# new continuous locations (available for adding)	
Switching	p=50	p=60
0 (starting)	47	57
1	10	26
2	16	9
3	11	8
4	8	4
5	4	6
6	0	7
7	-	5
8	-	5
9	-	4
10	-	2
11	-	6
12	-	0
$\sum$	96	139
Average	13.71	10.69

It can be observed that the number of the new added points was not necessarily decreasing. For instance, in the fourth switch when p=60 the number of the new points was 4 but, when these points were added, 6 other new points have been found in the next switch. A summary of Table 5.5 was displayed in Figure 5.12 (*a*) and Figure 5.12 (*b*) when p=50 and p=60 respectively.

Figure 5.12 (*a*): Number of new available continuous locations when p=50

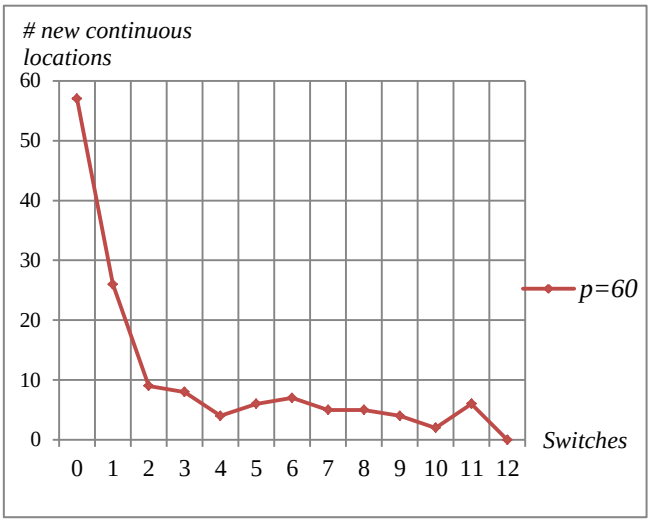


Figure 5.12 (*b*): Number of new available continuous locations when p=60

5.3.5 Computational results for RLS with different stopping conditions

The same data that was used earlier was also tested here. Table 6.6 shows the deviation values from the optimal solutions for RLS with the three different types of stopping conditions. We can conclude that the increase in the number of switches gave more chance to improve the solution, where the average values of switches were 2.3, 3.1 and 4.5 respectively and the average deviation values of these different stopping schemes were 10.39%, 10.38% and 10.21% respectively. In this study, we will use the latter stopping criterion as our stopping rule given its superiority. It can also be observed that the CPU time increased slightly with the number of switches where the average CPU times (in seconds) were 10.76, 12.54 and 16.55 respectively.

Table 5.6: Deviation (%) of the solutions from the optimal solution for the three stopping criteria

n = 439 TSP-Lib	The optimal solution (Z)	No improvement in the solution value Z				One extra iteration (a)				No new continuous facility configuration to add (b)			
p		Deviation (%)	# Switch-es	# Cplex Calls	CPU Time (sec)	Deviation (%)	# Switch-es	# Cplex Calls	CPU Time (sec)	Deviation (%)	# Switch-es	# Cplex Calls	CPU Time (sec)
10	1716.5099	5.95	1	24	6.41	5.95	2	35	8.89	5.95	2	35	9.26
20	1029.7148	1.76	2	35	9.65	1.76	2	35	9.64	1.76	2	35	10.04
30	739.19297	10.11	1	22	9.05	10.11	2	31	9.74	10.11	5	58	17.08
40	580.00539	7.94	2	29	9.60	7.94	3	38	12.24	7.94	3	38	13.58
50	468.54162	9.13	4	47	15.22	8.96	5	57	18.31	7.41	6	69	24.32
60	400.19527	9.15	3	42	13.51	9.15	3	35	12.48	9.15	12	16	33.24
70	357.94553	14.11	3	37	13.81	14.11	4	46	16.14	14.11	5	55	20.24
80	312.5	18.93	3	38	11.94	18.93	4	45	13.76	18.82	4	45	14.14
90	280.90256	14.84	2	26	9.23	14.84	3	34	13.28	14.84	3	34	12.15
100	256.68019	12.01	2	24	9.21	12.01	3	30	10.93	12.01	3	30	11.44
Average	614.21882	10.39	2.30	32.40	10.76	10.38	3.10	38.6	12.54	**10.21**	4.50	41.50	16.55

5.4 Enhancements on the RLS-based heuristic

In the original RLS algorithm all the new continuous locations are introduced as additional potential sites when solving the discrete problem. It is worth noting that for the *p*-centre problem it is not necessary that all the continuous locations need to be added in one step as in the multi-source Weber problem due to the property of the objective function. If we choose appropriately some of them in each single switch, this will increase the number of switches, which could provide more chance to improve the solution. Here, we propose two schemes that take this information into account. In the first one (RLS-Enh 1), we add one new continuous location only to the problem as a potential site whereas in the second (RLS-Enh 2), we use the structure of VNS when adding the new continuous locations (i.e., the number to add will increase if the solution does not improve).

Let us recall $X = (X_1, ..., X_p)$ to denote the location of the p facilities found in the continuous space with X_1 the centre of the largest circle and X_j the centre of the j^{th} nearest circle to X_1 based on the distance measure $d(X_1, X_j)$; $j = 2, ..., p$. For more information on the notation and the way of the covering circle CC_j are defined, see subsection 3.3.2.

Let P'_k be the k^{th} point to be added as potential site; $k = 1, ..., K_{\max} (K_{\max} \leq p)$.

5.4.1 Adding one continuous location only (RLS-Enh 1)

As there are several ways to add one new continuous location (k = 1) at each step, in this subsection, we propose three selection schemes. We refer to this as RLS-Enh 1.

a) A random selection- Here, in each switch, one continuous location is selected randomly using a uniform distribution from the obtained continuous locations. In other words, $P_1' \in \{X_1,..., X_p\}$ is chosen with probability $1/p$.

b) The facility location of the largest circle- In this scheme, the facility location which is the centre of the largest circle is selected. In the case of a tie, one of the facilities of these competing largest circles is chosen randomly. The idea is that the inclusion of such an important location will guide the solution when solving the discrete problem. Here, $P_1' = X_1$.

c) The location of the nearest facility to the largest circle- In each switch, the facility location sited at the centre of the nearest circle to the largest circle is selected. The reasoning behind this is similar to (*b*) except that here some demand points that are enclosed in the largest circle could be reallocated to the new one which in turn may reduce the radius of the largest circle. Here, $P_1' = X_2$.

Computational results of the three selection rules

Table 5.7 shows the results of the three schemes. This contains the deviation (%) of the solutions from the optimal solution, the CPU Time (sec), the number of switches and the number of Cplex calls. It is worth noting that the scheme that achieved the largest number of switches yielded better results than the others. For instance, schemes (*a*), (*b*) and (*c*) produced an average number of switches of 135.20, 136.80 and 115.10 times respectively while their respective average deviation values from the optimal solutions (in %) were 8.13, 8.12 and 9.01 respectively.

This was also true for the number of Cplex calls and the total CPU time consumed where the averages of the total CPU time were 349.74, 383.17 and 336.61 (sec) respectively. In general, we can say that the performance of the second scheme (*b*) was slightly better than the others. However, it is worth noting that this scheme did not always yield better results than its counterparts as shown when p = 30 and 60. It can also be noted that this enhancement (RLS-

Enh 1) produced a relatively larger number of switches compared to the best implementation of the original RLS procedure besides it yielded a better average results (8.12% vs 10.21%), see Tables 5.6 and 5.7.

Table 5.7: Deviation (%) of the solutions for three selection rules

n = 439 TSP-Lib	The optimal solution (Z)	Selection (a)				Selection (b)				Selection (c)			
p		Deviation (%)	# Switch-es	# Cplex Calls	CPU Time (sec)	Deviation (%)	# Switch-es	# Cplex Calls	CPU Time (sec)	Deviation (%)	# Switch-es	# Cplex Calls	CPU Time (sec)
10	1716.5099	5.95	13	158	38.85	5.95	10	126	31.63	5.95	10	123	30.77
20	1029.7148	0	45	500	130.39	0	32	341	72.15	0	44	494	125.13
30	739.19297	6.48	38	371	100.02	9.34	47	442	100.45	6.64	73	696	205.69
40	580.00539	7.20	63	545	175.99	7.37	74	616	194.90	7.37	72	593	193.03
50	468.54162	11.24	103	903	227.73	8.04	99	877	279.85	8.04	98	849	260.30
60	400.19527	6.38	185	1758	549.55	10.19	166	1494	417.59	16.70	146	1138	423.35
70	357.94553	9.32	197	1710	556.01	14.83	122	1048	303.82	9.10	222	2061	668.91
80	312.5	18.31	157	1221	376.58	11.53	318	2951	1016.67	15.95	145	1098	355.24
90	280.90256	8.64	332	1841	674.97	6.148	322	2796	943.09	10.47	121	928	350.99
100	256.68019	7.80	219	1715	667.32	7.80	178	1374	471.57	9.87	220	1678	752.67
Average	614.21882	8.13	135.20	1072.20	349.74	**8.12**	136.80	1206.50	383.17	9.01	115.10	965.80	336.61

5.4.2 Adding *k* continuous locations within a VNS structure (RLS-Enh 2)

Here, the number of the new continuous locations that are added as potential sites (say k) varies in the range $(1, K_{\max})$ instead of being fixed to 1 (as in RLS-Enh 1) or p (as performed in the original RLS). We set $K_{\max} = p$ to represent all the new found continuous locations. Furthermore, the structure of VNS is used where the value of k increased by one when there is no improvement, otherwise the value of k reverts back to $k = 1$. In brief, we start by adding one continuous location ($k = 1$) to the problem as a potential site, the new discrete location problem is then solved. If the new solution is improved, we revert back to $k = 1$, otherwise we explore adding $k+1$ new continuous locations. This process continues until $k = K_{\max}$. We refer to this enhancement as RLS-Enh 2. Three selection schemes are proposed for the addition these k new points:

a) A random selection- The k points are chosen randomly from the new obtained continuous locations, $P'_k \in \{X_1, ..., X_p\}; k = 1, ..., K_{\max}$.

b) Selection based on the location of the larger circles- The k points are chosen as the locations sited at the centre of the first k largest circles. In mathematical terms $P'_1 = X_1$ and $P'_k = X_{j'}$, $k=2,\ldots,K_{\max}$ with $j' = Arg \underset{\substack{j=2,\ldots p \\ X_j \neq P_S; S=1,..k-1}}{Max} R_j$

c) Selection based on the nearest circles to the largest circle- The first location is chosen as the centre of the largest circle and the remaining $k-1$ locations are taken as the centres of the nearest $k-1$ facilities to the largest circle. Here, we set $P'_t = X_t; t = 1,...,k; k = 1,...,K_{\max}$.

Computational results for the three selection rules for RLS-Enh 2

Table 5.8 shows the same information that was presented in Table 5.7. Here, it can also be observed that the scheme that achieved the largest number of switches yielded better results and consumes relatively more time than the others. The selection schemes (*a*), (*b*) and (*c*) yielded an average number of switches of 24.10, 28.50 and 25.30 respectively. Their respective average deviation values were 8.65%, 8.10% and 8.50% and their average total CPU times were 70.89, 86.12 and 75.08 (sec) respectively. Here, scheme (*b*) also outperformed (*a*) and (*c*) as shown in the previous experiment. However, the performance of (*a*) was found to be even worse than the one found in RLS-Enh 1 (*a*), while the percentage improvement in the performance of scheme (*c*) was the highest. We can also confirm that scheme (*b*) did not always yield better results than the other schemes, as shown in the case of $p = 60$ and 70.

It is worth noting that the performance of RLS-Enh 2 was slightly better than the one by RLS-Enh 1 in (*b*) and (*c*) except in scheme (*a*). However RLS-Enh 2 has reduced the number of switches significantly by around 80%. For instance the average number of switches fell from 136.80 to 28.50, which led to a reduction in the average total CPU time by around 78% (i.e., from 383.17 to 86.12 seconds), see Tables 5.7 and 5.8. Given that RLS-Enh 2 improved the performance of the original RLS algorithm, in the next subsection we will focus on introducing extra guidance on RLS-Enh 2 only while adopting the three selection rules.

Table 5.8: Deviation (%) of the solutions for the three selection rules (*a*, *b*, *c*)

n = 439 TSP-Lib	The optimal solution	Scheme (*a*)				Scheme (*b*)				Scheme (*c*)			
p	(*Z*)	Deviation (%)	# Switch-es	# Cplex Calls	CPU Time (sec)	Deviation (%)	# Switch-es	# Cplex Calls	CPU Time (sec)	Deviation (%)	# Switch-es	# Cplex Calls	CPU Time (sec)
10	1716.5099	5.95	6	81	20.12	5.95	4	57	15.64	5.95	7	94	22.84
20	1029.7148	0	11	134	36.76	0	11	125	36.71	0	10	124	38.39
30	739.19297	6.48	10	108	29.28	8.82	28	291	109.10	6.48	11	114	32.80
40	580.00539	6.04	22	207	64.19	7.37	13	113	39.44	7.20	18	166	60.79
50	468.54162	12.09	23	212	67.68	9.13	24	207	65.72	9.67	35	318	105.25
60	400.19527	6.38	35	327	99.15	10.19	28	262	77.21	9.37	26	231	74.29
70	357.94553	15.11	38	318	108.57	11.87	45	414	123.34	9.32	45	374	123.34
80	312.5	15.95	29	425	84.65	11.25	50	425	145.61	18.93	25	208	70.52
90	280.90256	8.64	42	337	112.90	7.54	49	407	144.31	8.64	42	338	115.61
100	256.68019	9.87	25	206	85.62	8.89	33	255	104.09	9.44	34	278	106.97
Average	614.21882	8.65	24.10	235.50	70.89	**8.10**	28.50	255.60	86.12	8.50	25.30	224.50	75.08

5.4.3 Guiding the search of RLS-Enh 2 via forbidden regions

In the previous subsections, all the new continuous locations were available for the selection to act as potential sites for the discrete problem. Given that the search is on the continuous space, some of these new continuous locations could be negligibly close to some of the potential sites and hence may not need to be considered. To achieve this, we restrict the choice by defining a small area around the new continuous locations as forbidden. In other words, any new continuous location which has any potential site in its forbidden region will be excluded from being selected. We aim to reduce the number of switches without affecting the quality of the solution.

Here, we define a forbidden region by the area enclosed by a circle with its centre as the new continuous location. In this study, the radius of each forbidden area is computed as a percentage of the radius of its original circle. We set $r_i = \beta R_i$ where r_i and R_i refer to the radius of the i^{th} forbidden circle and the radius of the i^{th} new found circle respectively. The parameter β is set close to zero say $\beta = 0.05$. Note that if the local search is applied on the discrete solution with p facilities, this generates $\bar{p}$ new continuous locations with $\bar{p} \leq p$ leading to $\bar{p}$ forbidden small circles. For instance, Figure 5.13 (*a*) shows an optimal solution of a *4*-centre discrete location problem (p_1, p_2, p_3 and p_4), whereas Figure 5.13 (*b*) shows a feasible continuous solution of the same problem ($\bar{p}_1$, $\bar{p}_2$, $\bar{p}_3$ and p_4). It can be noted that

applying the local search generates three new continuous locations say $\bar{p}_1, \bar{p}_2$ and p_3, with the fourth facility (p_4) remaining unchanged.

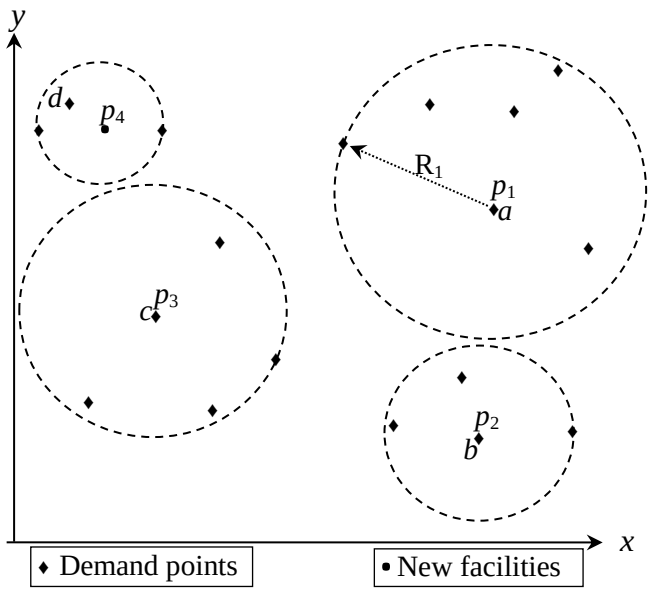


Figure 5.13 (*a*): An optimal solution of a *4*-centre discrete location problem

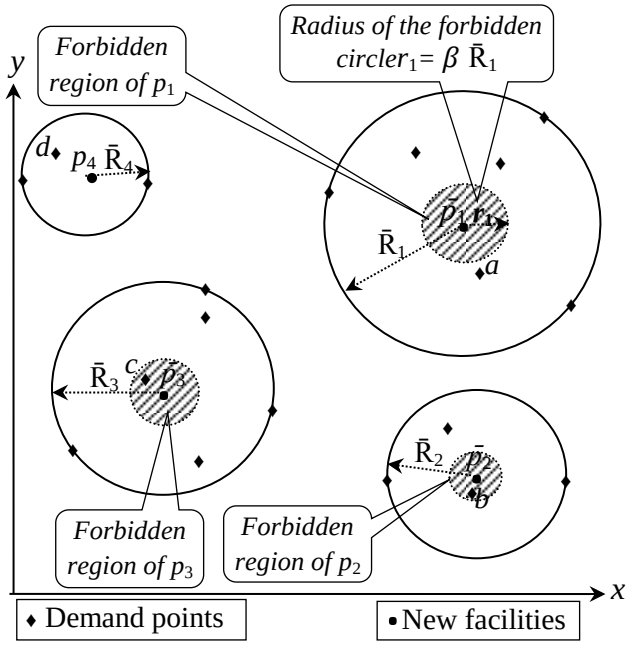


Figure 5.13 (*b*): A feasible solution of the same *4*-centre continuous location problem

Three forbidden regions based on $\bar{p}_1, \bar{p}_2$ and $\bar{p}_3$ are then produced. In this case, only one point ($\bar{p}_1$) can be chosen as potential site, while the others ($\bar{p}_2$ and $\bar{p}_3$) are rejected, as there are already potential sites namely *c* and *b* in these forbidden regions.

Computational results

The same n = 439 TSP-Lib data set was used here. In general, we can confirm that the introduction of forbidden regions improved the solution for all the three schemes (*a, b, c*). The average deviations without forbidden regions were 8.65%, 8.10% and 8.50% (as shown in Table 5.8) while with forbidden regions the figures reduced to 8.26%, 7.92% and 7.37% respectively as shown in Table 5.9. In addition, the average number of switches for all the three schemes has decreased to 21.40, 23.50 and 24.80 respectively. This has led to a decrease in the average total CPU times to 64.30, 68.09 and 73.23 (sec) respectively.

It is worth noting that the use of forbidden regions had relatively more effect on scheme (*c*). For example, without using forbidden regions, the performance of scheme (*b*) was better than the one of (*c*) (i.e., 8.10% vs 8.50% respectively, see Table 5.8), however, using

forbidden regions scheme (*c*) outperformed (*b*) (i.e., 7.37% and 7.92% respectively). Furthermore, the former was able to find the optimal solution twice out of 10 (when p =20 and p =30) whereas the others achieved the optimal solution once only (when p =20).

Table 5.9: Effect of forbidden region on the three selection rules (*a*, *b*, *c*) of RLS-Enh 2 with $\beta = 0.05$

n = 439 TSP-Lib	The optimal solution (Z)	Selection (*a*)				Selection (*b*)				Selection (*c*)			
p		Deviation (%)	# Switch-es	# Cplex Calls	CPU Time (sec)	Deviation (%)	# Switch-es	# Cplex Calls	CPU Time (sec)	Deviation (%)	# Switch-es	# Cplex Calls	CPU Time (sec)
10	1716.5099	5.95	4	57	14.45	5.95	4	59	14.44	5.95	6	80	19.91
20	1029.7148	0	10	121	32.61	0	10	110	32.52	0	10	118	30.63
30	739.19297	6.48	10	105	38.17	6.75	13	138	43.19	0	14	143	39.41
40	580.00539	7.37	14	136	42.62	7.00	20	185	61.72	6.04	19	178	61.15
50	468.54162	9.13	29	262	92.15	9.77	23	201	56.56	8.04	38	350	117.47
60	400.19527	7.88	28	239	76.12	10.19	26	247	78.55	9.15	28	251	74.35
70	357.94553	9.32	33	294	95.96	16.24	27	246	80.64	9.10	43	366	126.14
80	312.5	18.93	21	177	56.97	9.53	38	320	105.79	18.93	24	196	62.75
90	280.90256	8.64	29	215	80.67	6.15	37	294	104.07	8.64	30	226	82.97
100	256.68019	8.89	36	288	113.32	7.58	37	293	103.39	7.80	36	304	117.54
Average	614.21882	8.26	21.40	189.40	64.30	7.92	23.50	209.30	68.09	**7.37**	24.80	221.20	73.23

The forbidden regions were also tested with other values of β. It was observed that when β was large, too many new continuous locations were removed from further testing which made the search converges prematurely. The detailed results were not given here but were found to be similar to those presented in Table 5.9, see Appendix *D* for more details.

5.5 RLS extensions

The search seems to converge prematurely using a few switches only which goes against the RLS principle which relies on the use of a large number of shifts between the discrete and the continuous problems. One way to overcome this drawback is to extend the search by providing diversification through the injection of extra points as commonly used in meta-heuristics and particularly in population-based methods such as Genetic Algorithms (GA). The idea of injection was also used successfully for chromosomes injection in GA by Salhi and Gamal (2003) for a class of discrete location problems, and recently being investigated in Brimberg *et al*. (2013) for the multi-source Weber problem as part of RLS.

Strategies on how to generate the new injection points and the number of points that needs to be injected are presented next. For simplicity, we refer to RLS without injection using selection (c) as RLS1 and the one with injection as RLS2.

5.5.1 Injection strategies for the *p*-centre problem

Here, we propose two strategies which differ in the number of points that will be inserted in the problem. These include the following:

***a) The first strategy* (*F1*)**

Here, one point ($k=1$) is added randomly to the discrete problem. In other words, when RLS1 is stopped we inject one point randomly to the problem, with the aim to generate a new discrete optimal solution that would lead to new continuous points, triggering the process of switching. In subsequent iterations whenever the switching stops, we inject another point. This procedure is repeated until the following stopping condition is met. Here we consider the maximum allowed CPU time or the maximum number of injected points whichever is reached first.

***b) The second strategy* (*F2*)**

This strategy is similar to *F1* except that *k* new points ($k=1,..., K_{\max}$) are randomly added to the problem based on the VNS structure. In other words, when RLS1 stops, we inject one point and let RLS1 continues. If the solution is not improved, we increase the value of *k* by one ($k=k+1$), otherwise (the solution is improved) we continue with RLS1 until it stops where the injection process is triggered again with $k=1$. In this study we set $K_{\max} = \lceil \sqrt{p} \rceil$.

5.5.2 Empirical comparison of *F1* vs *F2*

An empirical comparison between *F1* and *F2* was conducted using the CPU time corresponding to 10,000 iterations of the Multi-Start Alternate Locate-Allocate Algorithm (MSALA) as our stopping condition. This setting as used in the two previous chapters was also used here throughout the remainder of the chapter. The results were summarised in Table 5.10. In general, we can confirm that the performance of *F2* was better than *F1* with

average deviations of 6.501% and 6.899% respectively. However, *F1* in some cases yielded better results than *F2*, as shown when p = 50 and p = 100. This is because *F1* can generate more switches than *F2* as shown by its average of 24.80 for RLS1 and 413.80 for RLS2 making a total of 438.60 switches, while *F2* generated 385.10 switches in total (24.80+ 360.30).

Table 5.10: Details of the solutions of RLS2 based on *F1* and *F2* strategies

n = 439 TSP-Lib	The optimal solution (Z)	The $F1$ Strategy (k = 1)						The $F2$ Strategy (k=1, 2,.. K_{max})					
p		Deviation (%)	# Switches			# Cplex Calls	CPU Time* (sec)	Deviation (%)	# Switches			# Cplex Calls	CPU Time* (sec)
			RLS 1	RLS 2	Total				RLS 1	RLS 2	Total		
10	1716.5099	5.95	6	142	148	1499	3.35	5.95	6	131	137	1499	3.36
20	1029.7148	0	10	136	146	1434	27.09	0	10	121	131	1308	29.13
30	739.19297	0	14	158	172	1573	26.02	0	14	131	145	1444	24.85
40	580.00539	7.37	19	343	362	4007	7.49	7.37	19	379	398	3927	8.33
50	468.54162	6.85	38	382	420	5011	366.70	8.04	38	464	502	4971	77
60	400.19527	6.38	28	613	641	6802	51.31	6.38	28	318	346	3640	51.38
70	357.94553	9.10	43	365	408	3999	97.62	9.10	43	434	477	4812	99.87
80	312.5	18.93	24	676	700	6486	10.96	13.21	24	586	610	5365	1631.34
90	280.90256	7.17	30	658	688	6087	123.03	7.17	30	508	538	5528	130.72
100	256.68019	7.25	36	665	701	5904	1334.70	7.80	36	531	567	5027	111.78
Average	614.21882	6.90	24.80	413.80	438.60	4280.20	204.83	**6.50**	24.80	360.30	385.10	3752.10	216.78

**: CPU time when the best solution is found, RLS1 refers to the case where no injection points are used.*

5.5.3 Enhancements on RLS2

As the second strategy (*F2*) is found to yield relatively better results than those found by *F1*, this led us to consider *F2* further by exploring two generation schemes for the k new points that are added as potential sites when solving the discrete problem. These schemes differ in the way the k extra points (injection points) are located randomly in the continuous space. These two strategies are explained in the next subsections.

a) Midpoints-based strategy (*F2a*)

The idea behind this strategy is to add the k extra points as a linear combination of the facility sited at the centre of the largest circle (i.e., located at X_1) and the other k nearest facilities to it located at $X_j; j = 2,...,k$. Let P'_k be defined as follows:

$$P'_k = \gamma X_1 + (1-\gamma) X_k \text{ with } \gamma \in (0,1); k = 2,..., K_{\max}$$

In particular for simplicity we use $\gamma = 1/2$, adopting a midpoint strategy which is similar to the one initially attempted by Brimberg *et al*. (2014) for the multi-source Weber problem. This result in the generation of the following new k points:

$$P'_k = \frac{X_1 + X_k}{2}; k = 1,..., K_{\max}$$

b) Covering circle-based strategy (F2b)

Here, the extra k points are added randomly in the k^{th} covering circle CC_k whose definition is defined in subsection 3.3.2. In other words, $P'_k \in CC_k; k = 1,..., K_{\max}$.

5.5.4 Empirical comparison of *F2a* and *F2b* for RLS2

An empirical comparison between *F2a* and *F2b* was performed using the same setting as before. According to Table 5.11, it can be observed that the performance of the second strategy (*F2b*) was better than (*F2a*) where the average deviations were 5.95% and 6.67% respectively. However, *F2b* did not always yield better results than *F2a*, as shown when p = 70. It can be noted that both strategies almost generated the same number of switches with a total average of 333.90 and 324.90 respectively.

Table 5.11: Details of the solutions of RLS2 based on *F2a* and *F2b* strategies

n = 439 TSP-Lib	The optimal solution (Z)	Midpoints strategy ($F2a$)						Covering circle strategy ($F2b$)					
p		Deviation (%)	# Switches			# Cplex Calls	CPU Time* (sec)	Deviation (%)	# Switches			# Cplex Calls	CPU Time* (sec)
			RLS 1	RLS 2	Total				RLS 1	RLS 2	Total		
10	1716.5099	5.95	6	91	97	1108	2.54	4.56	6	125	131	1529	321.06
20	1029.7148	0	10	55	65	2434	27.32	0	10	107	117	1299	26.51
30	739.19297	0	14	177	191	1894	29.43	0	14	119	133	1511	26.23
40	580.00539	6.04	19	53	72	684	88.88	5.91	19	247	266	3051	547.97
50	468.54162	8.04	38	274	312	3076	112.61	6.85	38	295	333	4061	559.97
60	400.19527	6.38	28	370	398	4204	203.37	6.11	28	444	472	5339	459.42
70	357.94553	6.61	43	376	419	4204	2080.15	9.10	43	349	392	4590	115.42
80	312.5	18.93	24	467	491	4844	10.82	15.95	24	406	430	4368	609.92
90	280.90256	7.17	30	588	618	5495	127.60	3.79	30	504	534	5805	749.76
100	256.68019	7.58	36	550	586	4654	116.40	7.25	36	495	531	5887	160.24
Average	614.21882	6.67	24.8	300.10	324.90	3259.70	279.91	**5.95**	24.80	309.10	333.90	3744	357.65

**: CPU time when the best solution was found, RLS1 refers to the case where no injection of point was used.*

5.6 Controlling the size of the augmented discrete problem

As the number of injected points may render the size of the augmented problem too large for the vertex *p*-centre problem to be solved optimally within a reasonable amount of time, to overcome this handicap we aim to manage the size of the augmented problem as governed by the number of potential sites (i.e., set E) by keeping it within a reasonable level. One way would be to restrict $|E| < K$ where K is an appropriate threshold for which the vertex *p*-centre problem can easily be solved. In Brimberg *et al*. (2014) the size is controlled by systematically deleting the old potential sites and using the newly added ones. In this study, we provide guidance by managing the size of *E*, we call this managed RLS2 as "M-RLS" for short which-will be used as our proposed RLS version.

It is worth noting that the strategy of the covering circle (*F2b*) is used in M-RLS for adding the number of extra injected points, but with a change in the maximum number of added points. In *F2b*, the number of added points ($k = 1,\ldots,K_{\max}$ with $K_{\max} = \lceil \sqrt{p} \rceil$) is relatively small. These *k* points are added randomly in the continuous space encompassed by the covering circle CC_k, which means that the maximum level of the covering circle is $CC_{K_{\max}}$. Whereas, in M-RLS, we increase the maximum number of added point ($K_{\max}$) to be max(*p*,50). But the maximum level of the covering circle is still kept the same, where we refer to $CC_l, l = 1,...,l_{\max}$ with $l_{\max} = \lceil \sqrt{p} \rceil$. In this case, it can be noted that the number of added points (*k*) can be greater than the maximum level of the covering circle ($l_{\max}$). In this case, if $k > l_{\max}$ we revert back to $l = 1$.

In brief, we start to add temporarily one extra point randomly $(k = l = 1)$ to the problem in the covering circle (i.e., $P_1' \in CC_1$) and apply RLS1 (no injection of points is used). If the solution is improved we add permanently this new continuous point and start the add process again with *k*=1 and *l*=1. Otherwise we remove this point and move to explore the second level $(k = l = 2)$ by adding temporarily two points randomly in CC_2 instead as potential sites and apply RLS1. If the solution is improved we add permanently these two points and start again from the first level $(k = l = 1)$, else we exclude these 2 points and add temporarily three points $(k = 3)$ in CC_3 $(l = 3)$. When there is no improvement and we reach $l_{\max} = \lceil \sqrt{p} \rceil$ we revert

back to $l = 1$ but the number of added points continues increasing until $k = K_{max} = \max(p,50)$ where this reverts back to k=1. The process continues, but if there is an improvement we start again from the first level, namely we set $k = l = 1$. In other words, the number of potential sites increases by k points only when there is an improvement in the solution.

5.7 Computational results

The same data set as tested earlier, was used to evaluate the performance of M-RLS. For the small data set (n=439), we compared the results of M-RLS to the optimal solutions provided by Chen and Chen (2009). For the other larger data sets where no optimal solutions are available, the overall best solutions found in the previous chapters were used instead. The same stopping condition that was used in previous chapters (i.e., the CPU for 10,000 iterations of MSALA) was also adopted here.

5.7.1 Comparisons against optimal results (small data set n=439)

In general, we can say that M-RLS gave encouraging results when compared to the other strategies (*F1'* and *F2'*). It is worth noting that adding a large number of extra injected points in the problem (M-RLS) has clearly improved the overall deviation (%) from 5.95 to 3.74%, as shown in Table 5.11 and Table 5.12 respectively.

Table 5.12: Solutions of M-RLS for small data set

n = 439 TSP-Lib	The optimal solution (Z)	M-RLS					
p		Deviation %	# Switches			# Cplex Calls	CPU Time (sec)
			RLS1	RLS2	Total		
10	1716.5099	2.29	6	61	67	1270	179.36
20	1029.7148	0	10	70	80	966	26.24
30	739.19297	0	14	71	85	1081	26.55
40	580.00539	2.11	19	103	122	2584	1119.84
50	468.54162	2.98	38	163	201	3853	1401.66
60	400.19527	3.64	28	199	227	4476	1175.27
70	357.94553	5.31	43	182	225	3542	1778.74
80	312.5	13.14*	24	196	220	3662	1522.68
90	280.90256	3.79	30	225	255	3473	346.75
100	256.68019	4.11	36	214	250	3726	1605.82
Average	614.21882	**3.74**	24.80	148.40	173.20	2863.30	918.29

*: *worst solution, RLS1 refers to the case where no injection of points was used.*

In addition, M-RLS found 2 optimal solutions out of 10, (i.e., when p =20 and p = 30). It can be noted that within M-RLS, RLS1 had a smaller number of switches than RLS2 with the average of 24.80 and 148.40 respectively.

5.7.2 Results on the larger data sets (*no optimal solution known*)

The larger datasets with no known optimal solutions (n= 575, 783, 1002 and 1323 TSP-Lib) were used to assess the performance of our proposed RLS namely M-RLS. For these large data sets we refered to the best known solutions found by the VNS-based heuristic (VNS2 (*FNV4*)) with learning) and the perturbation-based heuristic (STRONGPERT with learning).

In this section, we assessed the performance of M-RLS using the deviation from the best solution which was computed as Deviation (%) = $100\frac{Z_H - Z_{best}}{Z_{best}}$ with Z_{H} denoting the Z value found by heuristic 'H' and Z_{best} is the best known value of Z.

The results were shown in Table 5.13. It was found that the performances of both VNS and perturbation were relatively better than M-RLS, as the overall average deviation values from the best solutions were 0.61%, 0.90% and 5.38% respectively. It is worth noting that the performance of VNS2 (*FNV4*) and STRONGPERT were better than the performance of M-RLS. One of the main reasons was that in some cases, especially when n =783, the optimal discrete solution required an excessive amount of time compared to the other cases, which acts as a bottleneck for the search.

For instance, when n =783 and p=40, 50 and 60, the CPU time for 10,000 runs of the MSALA was found to be not enough even to find the optimal discrete solution, where their deviation values were 11.37% , 11.25% and 23.55% respectively. This is obviously the reason why in this case (n= 783 and p=40 and 50) the number of switches was recorded as zero. Furthermore, in some other cases (when p=80, 90) there was not enough time to explore the power of RLS2 (M-RLS). It may be interesting to use a powerful heuristic if necessary to replace the exact method. Another way would be to limit the time for the exact method and use the best feasible as the solution of discrete. Both ideas could be investigated in the future.

Table 5.13: Solutions of VNS, Perturbation and M-RLS for the larger data sets

n	*p*	Overall best solutions	VNS	Perturbation	M-RLS					
			VNS2 (*FNV4*) With Learning	STRONGPERT With Learning	Deviation %	# Switches			# Cplex Calls	CPU Time (sec)
						RLS 1	RLS 2	Total		
575	10	67.926	**0**	1.91	**0**	15	42	57	629	340.68
	20	45.622	**0**	**0**	1.99	21	7	28	290	930.70
	30	35.556	0.16	1.72	5.82	2	0	2	32	429.90
	40	30.265	**0**	0.49	6.31	9	0	9	101	241.25
	50	26.269	0.76	1.93	7.49	16	0	16	156	1589.55
	60	23.436	1.09	**0**	6.76	58	38	96	944	1410.98
	70	21.059	**0**	0.76	7.50	53	35	88	842	1138.74
	80	19.558	1.49	2.26	3.61	83	133	216	2041	894.33
	90	17.923	**0**	2.05	3.11	126	162	288	2647	1753.60
	100	16.697	0.01	1.86	3.93	143	225	368	3452	1222.62
	Average	30.431	**0.35**	1.30	4.65	52.60	64.20	116.80	1113.40	995.24
783	10	79.313	**0**	**0**	1.31	9	41	50	544	78.59
	20	53.461	**0**	2.13	2.70	14	3	17	174	523.37
	30	42.395	0.49	1.37	4.17	0	0	0	11	1942.01
	40	35.962	0.41	1.66	11.37+	0	0	0	5	2403.09
	50	31.357	0.17	**0**	11.25+	0	0	0	5	2514.47
	60	28.053	1.10	0.33	23.55+	0	0	0	2	2842.81
	70	25.446	0.69	2.16	9.63	0	0	0	10	3030.87
	80	23.560	**0**	0.44	10.72	6	0	6	61	949.08
	90	21.710	**0**	2.93	11.72	8	0	8	74	382.20
	100	20.334	**0**	2.34	5.05	139	74	213	2117	3334.01
	Average	36.159	**0.29**	1.34	9.15	17.60	11.80	29.40	300.30	1800.05
1002	10	2389.360	**0**	**0**	1.32	14	33	47	671	44.68
	20	1607.530	1.42	**0**	2.52	34	35	69	879	953.71
	30	1231.360	**0**	**0**	2.42	30	48	78	977	374.83
	40	1021.410	0.88	**0**	6.62	78	6	84	1031	904.15
	50	903.120	0.54	0.03	2.98	84	69	153	1894	2547.73
	60	795.709	0.73	0.59	5.62	96	89	185	2501	1927.72
	70	725.431	0.24	**0**	3.39	256	135	391	5479	3872.99
	80	660.019	1.78	0.14	1.99	263	175	438	6154	3520.03
	90	604.494	0.75	0.97	3.39	278	228	506	6894	1012.92
	100	564.795	1.03	0.21	4.47	222	204	426	5670	2765.85
	Average	1050.323	0.74	**0.19**	3.47	135.5	102.20	237.70	3215	1792.46
1323	10	2899.420	**0**	**0**	1.38	16	22	38	570	171.88
	20	1868.920	0.96	1.15	2.68	19	28	47	680	1083.13
	30	1477.590	0.26	0.88	3.96	21	4	25	302	2072.54
	40	1236.380	1.21	1.05	4.14	6	0	6	88	3965.57
	50	1060.820	0.42	0.68	4.14	24	1	25	325	3109.24
	60	940.691	0.13	**0**	8.26	11	0	11	129	5499.72
	70	844.967	**0**	2.05	3.66	102	72	174	2328	6431.40
	80	774.764	**0**	1.82	5.57	90	135	225	2910	2603.36
	90	719.580	2.27	**0**	2.02	165	147	312	3995	8375.45
	100	663.035	5.11	**0**	6.59	47	309	356	4946	329.03
	Average	1248.617	1.04	**0.76**	4.24	50.10	71.80	121.90	1627.30	3364.13
Overall Average		591.382	**0.61**	0.90	5.38	63.95	62.50	126.45	1564	1987.97

+: *Optimal discrete not guaranteed due to CPU time being larger than* $CPU_{max.,}$ *hence STRONG-RLS not used.*

5.7.3 Post-optimality results

For benchmarking purposes one way would be to combine the results of the approaches, namely VNS-based heuristics, the perturbation-based heuristics and STRONG-RLS, by using RLS after VNS (case 1) and RLS after the perturbation (case 2). In other words, we wish to see if STRONG-RLS can improve the best continuous solution of VNS and the perturbation. These implementations would obviously require extra computing time but the obtained results could be used to see if the additional effort is worthwhile and whether better results could be produced for benchmarking purposes.

For simplicity, we ran the perturbation and VNS for the same time as the time recorded by the multi-start procedure with 10,000 iterations say CPU_{max}. The stopping condition for M-RLS was either CPU_{max} or the number of extra injected points (K_{max}) was reached in 10 successive iterations without improvement whichever comes first. Here, the deviations (%) of these strategies (improvement) were based on the solutions of VNS or the perturbation.

$$Deviation\ (\%) = \frac{(Z_H - Z_C)}{Z_H}.100$$

where Z_H refers to the solution of a given heuristic '*H*' where '*H*' refers to VNS2 (*FNV4*) and STRONGPERT and Z_C denotes the value of *Z* found by case 1 and case 2. The five datasets (*n*= 439, 575, 783, 1002 and 1323) were used to test these strategies. In case 1, RLS improved the VNS2 (*FNV4*) solutions twice and the STRONGPERT solutions once, see Table 5.14.

5.8 Summary

In this chapter, we first presented the use of the optimal discrete solution as an initial solution when solving the continuous problem. The RLS procedure was reviewed and adapted to solve the *p*-centre location problem which was then enhanced by using the new discrete solution (*Z*) continuously as an upper bound for the next switch. Furthermore, we proposed two enhancements for RLS by incorporating forbidden regions. Extensions of RLS that cater for the introduction of injection points to provide diversification and avoid early stagnation alongside the management of the memory to control the size of the discrete problem were also explored.

Table 5.14: Post-optimality results

n	*p*	VNS + M-RLS (case 1)						perturbation + M-RLS (case 2)					
		VNS2 (*FNV4*) With Learning	M-RLS					STRONGPERT With Learning	M-RLS				
			Z	Improv-ement %	CPU time				Z	Improv-ement %	CPU time		
					Best	RLS1	RLS2				Best	RLS1	RLS2
439	10	1716.510	1716.510	0	0	4.64	431.41	1716.510	1716.510	0	0	4.42	431.67
	20	1029.710	1029.710	0	0	7.88	742.45	1029.710	1029.710	0	0	10.13	742.99
	30	739.193	739.193	0	0	8.42	1013.57	739.193	739.193	0	0	6.65	1017.93
	40	580.005	580.005	0	0	15.37	1152.76	580.005	580.005	0	0	12	1162.24
	50	471.699	471.699	0	0	5.91	1725.15	474.088	474.088	0	0	4.54	1726.12
	60	401.591	401.591	0	0	4.53	1979.67	**401.591**	**401.191**	**0.10**	**137.17**	4.70	1979.88
	70	357.946	357.946	0	0	11.58	2079.79	357.946	357.946	0	0	12.53	2075.77
	80	312.552	312.552	0	0	5.16	1932.90	315.486	315.486	0	0	10.33	1932.98
	90	282.013	282.013	0	0	4.44	1983.70	282.013	282.013	0	0	8.58	1980.38
	100	258.979	258.979	0	0	9.14	1852.16	257.570	257.570	0	0	11.72	1857.01
	Average	615.020	615.020	0	0	7.71	1489.36	615.411	615.372	0.01	13.72	8.56	1490.70
575	10	67.926	67.926	0	0	5.87	536.83	69.223	69.223	0	0	35.41	508.11
	20	45.622	45.622	0	0	51.70	892.94	45.622	45.622	0	0	97.62	847.06
	30	35.612	35.612	0	0	573.75	617.41	36.169	36.169	0	0	45.50	1146
	40	30.265	30.265	0	0	320.29	1118.76	30.414	30.414	0	0	108.62	1329.44
	50	26.469	26.469	0	0	9.70	1655.36	26.774	26.774	0	0	225.81	1441.13
	60	23.691	23.691	0	0	9.59	1779.99	23.436	23.436	0	0	71.69	1718.89
	70	21.059	21.059	0	0	24.12	2119.98	21.219	21.219	0	0	18	2128.47
	80	19.849	19.849	0	0	22.01	2146.65	20	20	0	0	195.82	1972.49
	90	17.923	17.923	0	0	11.21	2297.73	18.290	18.290	0	0	22.22	2285.99
	100	16.698	16.698	0	0	6.79	2525.88	17.007	17.007	0	0	38.08	2493.95
	Average	30.511	30.511	0	0	103.50	1569.15	30.816	30.816	0	0	85.88	1587.15
783	10	79.313	79.313	0	0	21.18	889	79.313	79.313	0	0	36.48	873.55
	20	53.461	53.461	0	0	15.79	1540.22	54.601	54.601	0	0	28.19	1528.11
	30	42.604	42.604	0	0	202.64	1853.36	42.974	42.974	0	0	1646.68	409.75
	40	36.110	36.110	0	0	2405.21	0.12	36.560	36.560	0	0	2402.78	0.98
	50	31.409	31.409	0	0	1600.46	914.74	31.357	31.357	0	0	2059.51	456.31
	60	28.361	28.361	0	0	309.12	2533.98	28.147	28.147	0	0	920.21	1922.99
	70	25.622	25.622	0	0	36.26	3118.91	25.996	25.996	0	0	303.70	2852.03
	80	23.560	23.560	0	0	14.09	4452.55	23.665	23.665	0	0	18.53	4449
	90	21.710	21.710	0	0	21.48	3649.89	22.346	22.346	0	0	31.08	3617.41
	100	20.334	20.334	0	0	847.13	3228.76	20.809	20.809	0	0	638.28	3437.99
	Average	36.248	36.248	0	0	547.34	2218.15	36.577	36.577	0	0	808.54	1954.81
1002	10	2389.360	2389.360	0	0	10.44	938.39	2389.360	2389.360	0	0	10.55	939.10
	20	1630.300	1630.300	0	0	27.35	1599.99	1607.530	1607.530	0	0	14.12	1614.56
	30	1231.360	1231.360	0	0	19.54	2543.52	1231.360	1231.360	0	0	19.94	2543.54
	40	1030.400	1030.400	0	0	20.58	2939.63	1021.410	1021.410	0	0	16.78	2944.01
	50	907.980	907.980	0	0	7.25	3676.63	903.399	903.399	0	0	21.97	3662.31
	60	801.474	801.474	0	0	19.20	4502.50	800.391	800.391	0	0	20.65	4502.07
	70	727.154	727.154	0	0	26.38	6614.81	725.431	725.431	0	0	19.02	6622.88
	80	671.751	671.751	0	0	21.48	7005.71	660.913	660.913	0	0	23.08	7004.51
	90	608.999	608.999	0	0	20.76	6863.39	610.328	610.328	0	0	51.52	6833.95
	100	570.636	570.636	0	0	39.08	7093.43	565.962	565.962	0	0	29.89	7103.22
	Average	1056.941	1056.941	0	0	21.21	4377.80	1051.608	1051.608	0	0	22.75	4377.01
1323	10	2899.420	2899.420	0	0	34.11	1551.71	2899.420	2899.420	0	0	31.79	1555.34
	20	1886.820	1886.820	0	0	200.67	2239.70	1890.430	1890.430	0	0	250.97	2191.01
	30	1481.400	1481.400	0	0	33.15	3423.01	1490.640	1490.640	0	0	27.64	3428.15
	40	1251.340	1251.340	0	0	50.90	4043.55	1249.300	1249.300	0	0	36.23	4058.52
	50	1065.280	1065.280	0	0	91.60	5587.13	1068.040	1068.040	0	0	151.70	5526.99
	60	941.870	941.870	0	0	109.25	6419.92	940.691	940.691	0	0	77.16	6454.08
	70	844.967	844.967	0	0	100.15	7417.74	862.269	862.269	0	0	89.40	7427.35
	80	774.764	774.764	0	0	149.83	7757.81	788.885	788.885	0	0	127.44	7778.77
	90	**735.878**	**734.371**	**0.21**	**226.89**	59.92	8359.10	719.580	719.580	0	0	35.53	8383.32
	100	**696.938**	**696.317**	**0.09**	**1709.72**	170.83	8846	663.035	663.035	0	0	31.54	8984.88
	Average	1257.868	1257.655	0.03	193.66	100.04	5564.57	1257.229	1257.229	0	0	85.94	5578.84
Overall Average		599.318	599.275	0.01	38.73	155.96	3043.81	598.328	598.320	0.00	2.74	202.33	2997.70

The computational results showed that a significant improvement compared to the original implementation was found. For instance, the average deviation (%) when n =439 has decreased by nearly half from 7.37% to 3.74%.

However, the best enhancements for VNS2 (VNS2 (*FNV4*) with learning) and the perturbation (STRONGPERT with learning) still outperformed the best RLS. One of the reasons was because in some data sets (not necessarily the largest, such as n =783) finding the optimal solution for the discrete problem consumed a relatively long time, leaving a negligible or no time to be used for further exploration of the search. The use of STRONG-RLS was also tested after VNS2 (*FNV4*) and STRONGPERT to see if better results could be found for benchmarking purposes. However, in our experiments only three new better solutions were discovered. It is worth pointing at that the overall conclusion was not that disappointing given that the RLS concept is very new and future implementations will hopefully enhance its position as one of the major performers in global optimisation.

Chapter 6

Conclusion and Suggestions

The aims of this chapter are to provide a summary of the proposed approaches and the outcomes that have been found in this thesis followed by the research areas that could be explored in the future.

6.1 Conclusion

In this thesis, the *p*-centre location problem in the continuous space was investigated. Firstly, we proposed simple but effective enhancements on the original Elzinga-Hearn algorithm for both the weighted and the unweighted *1*-centre problem on the plane. A Variable Neighbourhood Search heuristic (VNS) using both the customer-based as well as the facility-based neighbourhood types was proposed to solve this related location problem. Furthermore, two effective enhancements on our local search (Cooper's approach) were designed followed by a scheme that incorporates learning within the search. Two types of perturbation-based heuristic were proposed followed by efficient enhancements and a learning scheme. In this approach, we allowed the amount of perturbations to be variable. Finally, the new concept of Reformulation Local Search (RLS) which shifts between solving the continuous problem and the augmented discrete problem and which was originally designed for the multi-source Weber problem was adapted to solve this location problem. This was followed by a scheme for generating a tighter new upper bound when solving the discrete problem. Two enhancements on the RLS procedure that use forbidden regions were also proposed. Extensions of RLS were provided including the use of injection points for diversification and memory management at the discrete phase. Our approaches on the existing datasets (n = 439, 575, 783, 1002 and 1323 TSP-Lib) were used with encouraging results.

Chapter 1 concentrated on the literature review on the *p*-centre problem in the continuous space. A description of the problem under study and a motivation for this research was presented followed by applications and a brief description of a possible classification of location problem. To our knowledge there were only constructive heuristics and no meta-

heuristic for this problem. Also it was noted that small to medium size instances were used only. Our study aimed to fill those gaps. The meta-heuristics methods that have been used in this research were also covered in the first chapter. The single facility minmax location problem (*1*-centre) and the multi-facility minmax location problem (*p*-centre) were then reviewed here including a brief description of the exact method used to solve the vertex *p*-centre problem as this has been incorporated into our search when solving the continuous case.

In the second chapter, the Elzinga-Hearn algorithm (*1*-centre) for both the unweighted and the weighted case was described first. This was followed by a highlight of the weakness of this method which gave rise to the design of our enhancements. The idea behind these enhancements was to determine the best points that can be used as initial starting points for the algorithm as well as the best uncovered point (the point that is outside the current circle) to be the third point for the current solution. In our experiments, we noted that the best two enhancements (*V3* and *V4* for the unweighted case and *W3* and *W*4 for the weighted case) outperformed the original algorithm (*V0*) for all values of *n*, whereas the other enhancements were found to be quicker than *V0* when $n < 30$ for the unweighted case and when $n < 50$ for the weighted case. To distinguish between the performance of these two best enhancements (*V3* vs *V4* and *W3* vs *W*4), additional tests using large value of *n* ($n = 50$ to 1000) were performed. In brief, our best enhancements yielded about 60% reduction in CPU time. In summary, we proposed interesting and effective rules that can be used when solving the *1*-centre problem as part of the *p*-centre problem. For illustration purpose, a simple multi-start approach was also used to show the effectiveness of these enhancements.

In the third chapter, a VNS that includes the Customer-based Neighbourhood (*CN*) and one using the Facility-based Neighbourhood (*FN*) were designed to solve this continuous location problem. The original heuristic of *FN* was applied in two cases by swapping the facilities with random fixed points (discrete space VNS1(*FN*)) and with points in the continuous space (VNS2(*FN*)). Several enhancements were then developed for VNS(*CN*) and for VNS2(*FN*). In addition, two improvement procedures on our local search (Cooper's approach) that use a critical point-based allocation and the removal of the non-promising circles were also proposed. Furthermore, a learning scheme based on previous iterations was incorporated into the search to identify the best area where to relocate

the open facilities using the levels of the covering circle. The computational results showed that incorporating learning within VNS2(*FNV4*) has clearly improved its initial performance to become the best performer. Using the CPU time for the Multi-Start algorithm of 10,000 runs on the data (n = 439 TSP-Lib), where optimal solutions are known, the VNS heuristic proved to be performing rather well. This was then applied in larger data sets with various value of p where no optimal solution is known.

In chapter 4, the perturbation heuristic that allows the search to be infeasible by allowing additional or fewer facilities, was presented. The original perturbation algorithm and the strong perturbation which we call "GRADPERT" and "STRONGPERT" respectively were proposed and adapted for this problem. Furthermore, a powerful enhancement, where the number of new facilities (q) that could go over and under the required number of facilities (p) was relaxed and made dynamic instead of being fixed was developed. This is enhanced by incorporating learning within the search. The computational experiments on the small data set (n=439) where optimal solutions exist and on the large datasets (n = 575, 783, 1002 and 1323), showed that when learning was incorporated within the search, the overall average deviation has clearly improved. Finally, the comparison between VNS and perturbation-based heuristic was also presented. In general, we can say that the performance of the VNS-based heuristic was slightly better than the perturbation-based heuristic, but in several cases the latter approach also yielded better solutions.

In the fifth chapter, we investigated the idea of using the optimal solution of the discrete case as an initial solution for the continuous problem. The new concept of reformulation local search (RLS) which aims to shift between the continuous problem and the augmented discrete problem, and which was originally designed for the multi-source Weber problem, was adapted here. A scheme for generating a tighter upper bound when iteratively solving optimally the discrete problem was also presented. In order to increase the number of switchings between the discrete and the continuous phase, different stopping conditions were considered. Here, we designed two strategies for adding the new continuous locations as potential sites to the discrete problem. The VNS structure that has a variable number of added points was shown to be the best. This is enhanced by the use of forbidden continuous regions, where the new continuous locations that were very close to the potential sites were excluded

from being selected. Extensions of the RLS were also provided including the use of injection points as potential sites for diversification and the management of memory at the discrete phase. This is performed to control the size of the discrete problem. The computational experiments showed that the best scheme for RLS1 yielded encouraging results, when enhanced by STRONG-RLS. However, it was found in general that the performances of both VNS and the perturbation were relatively much better than the proposed RLS scheme which is still, in our view, in its infancy.

6.2 Future research suggestions

In this section, we highlight some research areas that we believe may be worthwhile studying in the future. The suggestions for this research can be divided into two categories. This includes improving the performance of the meta-heuristics used in this study and adapting these meta-heuristics to solve other related *p*-centre problems.

6.2.1 Improving the performance of the used meta-heuristics

In this section, we proposed several ideas which can lead to possible improvements on our meta-heuristics. These concepts are summarised as follows:

Enhancing the local search

The local search that has been used in this study is similar to that of Cooper (1964), which is based on switching between the location and the allocation phase until no further improvement can be found. In the third chapter, this local search included two enhancements on the Elzinga-Hearn algorithm and another two on the allocation phase where a critical point-based allocation and the removal of the non-promising circles are used. However, this local search could be made more powerful by including a short meta-heuristic such as RLS1, a mini perturbation (STRONGPERT-V2) or a Variable Neighbourhood Descent (VND). As this may require extra computational time, challenging implementation issues need to be taken into account.

Enhancing the perturbation-based heuristics

In the method proposed in Chapter 4 (STRONGPERT-V2), the way the q facilities are dropped could be revisited. In our study, the first candidate facility for removal is the

facility that increases the objective function the least (q facilities are chosen from all the open facilities). This drop process can be modified to be similar to the add process, so the q facilities to remove could be chosen from the covering circle instead. This can be followed further by a scheme that incorporates learning within this drop process.

Enhancing the reformulation local search (RLS)

The suggestions for the RLS, which is a very recent local search, are organised into three categories which are as follows:

Guiding the search via forbidden regions and memory management

In our proposed method (RLS), the new continuous locations that are very close to the potential sites (located in forbidden regions) are excluded from the selection as potential sites. This guidance can be reflected by excluding the potential sites that are very close to the new continuous locations instead. This is because the new continuous locations can be more informative than the old potential sites. Also, the way we manage the size of the memory could be based on the frequency of occurrence of the potential sites measured by small representative circles. In brief, those sites that are seldom selected could be ignored making the size of the discrete problem more manageable to solve without losing solution quality.

Increasing the set of potential sites

The perturbation scheme allows us to generate many candidate "centres" during its up and down trajectories. These new centres could be used within the RLS framework to increase the set of potential sites and hence provide opportunities for determining better solutions. Hybrid algorithms based on perturbation methods and RLS could then be considered in the future to include such a view. As the size of the discrete problem may get too big some form of selection and memory management need to be considered.

Incorporating RLS within VNS2 (FNV4) and STRONGPERT-V2

Both VNS as well as the perturbation based-heuristic can incorporate RLS as part of their local search. One way would be to combine both approaches namely VNS2 (*FNV4*) and RLS. We can use the RLS after a full cycle of VNS2 (*FNV4*) terminates. As this hybrid realization may require an excessive amount of CPU, some guidance on when to call for such a scheme could be worth exploring.

6.2.2 Applying the used meta-heuristics for other *p*-centre related problem

In this study, we adapted three meta-heuristics to solve the classical *p*-centre problem on the plane. Here, we suggest other types for this problem which can be solved by modifying the enhanced versions of the meta-heuristics. The following three related *p*-centre problems are briefly given here.

The conditional p-centre problem

In the unconditional *p*-centre problem, we need to find the location of *p* new facilities to minimise the maximum distance between a customer and its nearest facility, whereas in the conditional *p*-centre problem, there are *q* locations of facilities which are given (*q* existing facilities). We need to find the location of *p* additional facilities to minimise the maximum distance between a customer and its nearest facility, whether new or existing facility. This model is important, because in real-world emergency systems, we need to locate additional facilities to improve their customer service levels. The conditional *p*-centre problem is studied by Berman and Simchi-Levi (1990), Chen and Handler (1993), Berman and Drezner (2008) and recently by Chen and Chen (2010).

The meta-heuristics that have been used in this study can be modified to solve this type of problem. For instance, we can define small areas around the given *q* existing facilities as forbidden. In other words, when we need to relocate the facilities (in VNS), adding facilities (in perturbation-based heuristic) and adding the injection points (in RLS), these facilities must not be located in these forbidden region.

The equality in service delivery

In the classical *p*-centre problem, the value of the solution is determined by the radius of the largest circle, while the radii of the other smaller circles may vary in size considerably. This model does not aim to achieve the equality in service delivery, although in some cases the equality in service is very important. For instance, the goals in the public sector are social cost minimization, equity and efficiency. The geometrical interpretation for this problem is to find the locations of *p* circles that have very similar radii to cover all the demand points, see Suzuki and Drezner (1996) and Ezra *et al.* (1994).

The α-neighbor p-centre problem

The possibility of providing service to the customer by multiple facilities is also useful when customer is required to withstand service facility failures. The importance of this model is increased in the areas that have the highest population density to ensure that service delivery to the customer is performed in a timely manner. This problem, presented by Krumke (1995) is a generalisation of the *p*-centre problem, such that every customer is assigned to α facilities and were the objective is to minimise the maximum distance between a customer and its α^{th} nearest facility. For more details, see Chen and Chen (2013) where a brief literature review on this problem was also given.

Bibliography:

[1] Ahuja R. K., Ergun, Ö, Orlin, J. B. and Punnen, A. B., 2002, "A survey of very large-scale neighborhood search techniques", *Discrete Applied Mathematics*, 123:75–102.

[2] Alharbi, A., 2010, "Combining Heuristic and Exact Approach for the vertex p-centre problem and other related location problems", *PhD Thesis*, Kent Business School, University of Kent, UK.

[3] Al-khedhairi, A. and Salhi, S., 2005, "Enhancement to Two Exact Algorithms for Solving the Vertex p-Center Problem", *Journal of Mathematical Modeling and Algorithms*, 4: 129-147.

[4] Berman, O. and Drezner Z., 2008, "A new formulation for the conditional *p*-median and *p*-center problems", *Operations Research Letters*, 36: 481-483.

[5] Berman, O. and Simchi-Levi, D., 1990, "Conditional location problems on networks", *Transportation Science*, 24: 77-78.

[6] Brimberg J, Drezner Z, Mladenović N and Salhi S. Some enhancements on the reformulation local search for continuous location problems, Presentation given at Spanish OR, Spring, June 2013.

[7] Brimberg J, Drezner Z, Mladenović N and Salhi S., 2014, "A new local search for continuous location problems", *European Journal of Operational Research*, 232: 256-265.

[8] Brimberg, J. and Mladenovic, N., 1996, "A Variable Neighbourhood Algorithm for Solving the Continuous Location-Allocation Problem", *Studies in Locational Analysis*, 10: 1-10.

[9] Brimberg, J., Hansen, P., Mladenovic, N. and Salhi, S., 2008, "A Survey of Solution Methods for the Continuous Location-Allocation Problem", *International Journal of Operations Research*, 5: 1-12.

[10] Brown, J. R., 1972, "Chromatic scheduling and the chromatic number problem ", *European Management Science*, 19 (4): 456-463.

[11] Burstall, R. M., Leaver, R. A. and Sussams, J. E., 1962, "Evaluation of transport costs for alternative factory sites - a case study", *Operational Research Quarterly* 13: 345-354.

[12] Carbone, R. and Mehrez, A., 1980, "The single facility minimax distance problem under stochastic location of demand", *Management Science* 26: 113-115.

[13] Chakraborty, R. K. and Chaudhuri, P. K., 1981, "Note on Geometrical Solution for Some Minimax Location Problems", *Transportation Science*, 15: 164-166.

[14] Chen, R., 1983, "Solution of Minisum and Minimax Location-Allocation Problems with Euclidean Distances", *Naval Research Logistics Quarterly*, 30: 449-459.

[15] Chen, D. and Chen, R., 2009, "New relaxation-based algorithms for the optimal solution of the continuous and discrete *p*-center problems", *Computers & Operations Research*, 36: 1646-1655.

[16] Chen, D. and Chen, R., 2010, "A relaxation-based algorithm for solving the conditional *p*-center problem", *Operations Research Letters*, 38: 215-217.

[17] Chen, D. and Chen, R., 2013, "Optimal algorithms for the α-neighbor *p*-center problem", *European Journal of Operational Research*, 225: 36-43.

[18] Chen, R. and Handler, G. Y., 1987, "Relaxation method for the solution of the minimax location-allocation problem in Euclidean space", *Naval Research Logistics*, 34: 775-788.

[19] Chen, R. and Handler, G. Y., 1993, " The conditional *p*-center problem in the plane", *Naval Research Logistics*, 40: 117-127

[20] Cooper, L., 1963, "Location-Allocation Problems", *Operations Research*, 11:331-343.

[21] Cooper, L., 1964, "Heuristic Methods for Location-Allocation Problems", *SIAM Review*, 6: 37-53.

[22] Chrystal, G., 1885, "On the problem to construct the minimum circle enclosing n given points in a plane", *Proceedings of the Edinburgh Mathematical Society, third meeting, January 9th*.

[23] Daskin, M., *Network and Discrete Location: Models, Algorithms and Applications*, John Wiley & Sons, Inc., New York, 1995.

[24] Daskin M., 2000, "A new approach to solving the vertex p-center problem to optimality: algorithm and computational results", *Communications of the Operations Research Society of Japan*, 45(9):428-436.

[25] Daskin, M., 2008, "What you should know about location", *Naval research Logistics*, 55(4): 283-294.

[26] Dongarra, J. J., Performance of various computers using standard linear Equation software. http://www.netlib.org/benchmark/performance.pdf (Accessed online 15 April 2013).

[27] Drezner, Z., 1984*a* "The Planar Two-Center and Two-Median Problems". *Transportation Science*, 18: 351-361.

[28] Drezner, Z., 1984*b*, "The p-center problem-heuristic and optimal algorithms", *Journal of the Operational Research Society*, 35:741-148.

[29] Drezner, Z., *Facility location: A Survey Application and Methods*, Springer, New York, 1995.

[30] Drezner, Z., and Hamacher, H., *Facility location: applications and theory*, Springer, New York, 2001.

[31] Drezner, Z. and Shelah, S, 1987, "On the complexity of the Elzinga-Hearn algorithm for 1-Center problem", *Mathematics of Operations Research*, 12 (2): 255-261.

[32] Dyer, M. E. and Frieze, A. M., 1985, "A Simple Heuristic for the p-Centre problem", *Operations Research Letters*, 3: 285-288.

[33] Eilon, S., Watson-Gandy, C.D.T., and Christofides, N., *Distribution Management: Mathematical Modelling and Practical Analysis*, Hafner, New York, 1971.

[34] Eiselt, H. A. and Charlesworth, G., 1986, "A Note on p-Center Problems in the Plane", *Transportation Science*, 20: 130-133.

[35] Eiselt, H. A. and Marianov, V., *Foundations of Location Analysis*, Springer, New York, 2011.

[36] Elloumi, S., Labbe, M. and Pochet, Y., 2004, "A New Formulation and Resolution Method for the p-Center Problem ", *INFORMS journal on Computing*, 16 (1): 84-94.

[37] Elzinga, J. and Hearn, W. D., 1972, "Geometrical Solutions for Some Minimax Location Problems", *Transportation Science*, 6: 379-394.

[38] Ezra, N., Handler, G. Y. and Chen, R., 1994, "Solving Infinite *p*-Center Problems in Euclidean Space Using an Interactive Graphical Method", *Location Science*, 2:

101-109.

[39] Gamal, M. and Salhi, S., 2001, "Costructive Heuristics for the Uncapacitated Continuous Locationt-Allocation Problem", *Journal of the Operational Research Society*, 52: 821-829.

[40] Gleason, J., 1975, "A set covering approach to bus stop location", *Omega*, 3, 605-608.

[41] Hakimi, S. L., 1964, "Optimum Location of Switching Centers and the Absolute Centers and Medians of a Graph", *Operations Research*, 12, 450-459

[42] Hakimi, S., 1965, "Optimum Distribution of Switching Centers in a Communication Network and Some Related Graph Theoretic Problems", *Operations Research*, 13: 462-475.

[43] Hanafi, S. and Freville, A., 1998, "An efficient tabu search approach for the 0-1 multidimensional knapsack problem", *European Journal of Operational Research*, 106: 659-675.

[44] Hamacher, H. W. and Klamroth, K., 2000, "Planar Weber location problems with barriers and block norms" *Annals of Operations Research*, 96: 191-208.

[45] Handler, G., and Mirchandani, P., *Location on Networks: Theory and Algorithms*, MIT Press, Cambridge, MA, 1979.

[46] Hansen, P. and Mladenovic, N., 1997, "Variable Neighbourhood Search", *Location Science*, 5: 207-225.

[47] Hansen, P., and Mladenovic, N., 2003, "Variable Neighbourhood Search" In Glover F. and Kochenberger G.A.(eds), Handbook of Meta-heuristics, Kluwer Academic Publisher, London, 145-184.

[48] Hansen P., Mladenovic N., Brimberg J. and Moreno-Perez JA. "Variable Neighbourhood Search" In Gendreau M and Potvin J-Y., (eds) Handbook of Meta-heuristics, chapter 3, 61-86, 2010. (2nd edition), Springer, NY.

[49] Hansen, P., Mladenovic, N., and Taillard, E., 1998, "Heuristic Solution of the Multisource Weber Problem as a *p*-Median Problem", *Operations Research Letters*, 22: 55-62.

[50] Hearn, D. W. and Vijay, J., 1982, "Efficient Algorithms for the (Weighted) Minimum Circle Problem", *Operations Research*, 30: 777-795.

[51] Hurtado, F., Sacristán, V. and Toussaint, G., 2000, "Some Constrained Minimax

and Maximin Location Problems", *Studies in Locational Analysis issue*, 15: 17-35.

[52] Ilhan, F. T., Özsoy, A. and Pinar, M. C., "An efficient exact algorithm for the vertex p-center problem and computational experiments for different set covering subproblems", Technical Report, 2002. Available at http://www.optimization-online.org/DB_HTML/2002/12/588.html.

[53] Ilhan, T. and Pinar, M., 2001, "An efficient exact algorithm for vertex p-center problem", online report, http://www.optimization- online.org/DB_HTML/ 2001/09/376.html

[54] Kariv, O., and Hakimi, S.L., 1979, "An Algorithmic Approach to Network Location Problem, I: The p-Centers", *SIAM Journal on Applied Mathematics*, 37: 539-560

[55] Klamroth, K., 2001, "A reduction result for location problems with polyhedral barriers" *European Journal of Operational Research*, 130: 486-497.

[56] Krumke, S. O., 1995, " On a generalization of the *p*-center problem", *Information Processing Letters*, 56: 67-71.

[57] Luis, M., 2008, "Metaheuritics for the Capacitated Multi-source Weber Problem", *PhD Thesis*, Kent Business School, University of Kent, UK.

[58] Luis, M., Salhi, S. and Nagy, G., 2011, "A guided reactive GRASP for the capacitated multi-source Weber problem", *Computers & Operations Research*, 38: 1014-1024.

[59] Maffioli, F. and Righini, G., 1994, "An Annealing Approach to Multi-Facility Location Problems in Euclidean Space", *Location Science*, 2, 205-222.

[60] Mansfield, E. and Wein, H. H., 1958, "A Model for the Location of a Railroad Classification Yard ", *Management Science*, 4 (3): 292-313.

[61] Megiddo, N., 1983, "The Weighted Euclidean 1-Center Problem" *Mathematics of Operations Research*, 8: 498-504.

[62] Megiddo, N. and Supowit K. J., 1984, "On the complexity of some common geometric location problems" *SIAM Journal of Computing*, 13: 182-196.

[63] Minieka, E., 1970, "The m-Center Problem ", *SIAM Review*, 12: 138-139.

[64] Mladenović, N., and Hansen, P., 1997, "Variable Neighborhood Search", *Computers and Operations Research*, 24 (11):1097-1100.

[65] Mladenović, N., Labbé, M. and Hansen, P., 2003, "Solving the p-Center Problem with Tabu Search and Variable Neighborhood Search", *Networks*, 42 (1): 48-64.

[66] Mladenović, N., Salhi, S., Hanafi, S. and Brimberg, j., 2014, "Advances in Variable Neigh" vol 52.

[67] Pelegrin, B. and Canovas, L., 1998, "A new assignment rule to improve seed points algorithms for the continuous k-center problem", *European Journal of Operational Research*, 104: 366-374.

[68] Pisinger, D. and Ropke, S., 2010, "Large Neighborhood Search", Handbook of Metaheuristics: International Series in Operations Research and Management Science, Springer, Berlin, 399-419.

[69] Plastria, F., Continuous covering location problem. *Facility Location: Applications and Theory* (Drezner, Z., ed.). Springer, New York, 2002, pp. 37-79.

[70] Plesnik, J., 1987, "A heuristic for the p-center problems in graphs", *Discrete Applied Mathematics*, 17: 263-268.

[71] Preparata, F., and Shamos, M., *Computational geometry an introduction*, Springer-Verlag, New York, 1985.

[72] Price, W.L., and Turcotte, M., 1986, "Locating a blood bank", *Interfaces*, 16: 17-26.

[73] Reeves, C., *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell, Oxford, 1993.

[74] ReVelle, C., Bigman, D., Schilling, D., Cohon, J. and Church, R., 1977, "Facility location: A review of context-free and EMS models", *Health Services Research*, 129-146.

[75] Righini, G., 1993, "A geometric algorithm for the Euclidean 1-center problem", *Giornate AIRO* 1993 (Capri).

[76] Saatcioglu, O., 1982, "Mathematical programming models for airport site selection", *Transportation Research*, Part B 16 (6), 435-447.

[77] Salhi, S., 1997, "A Perturbation Heuristic for a Class of Location Problems", *Journal of the Operational Research Society*, 48: 1233-1240.

[78] Salhi, S., "Heuristic Search Methods", In G.A. Marcoulides, editor, *Modern Methods for Business Research*, Lawrence Erlbaum Associates, New Jersy, 1998, 147-175.

[79] Salhi, S., 2006, "Heuristic Search In Action: The Science of Tomorrow", Paper presented at OR 48 Conference, University of Bath, Bath, UK, in keynote papers, (Salhi, S., ed.) pp. 39-58.

[80] Salhi, S., and Al-Khedhairi, A., 2010, "Integrating heuristic information into exact method: The case of the vertex *p*-centre problem", *Journal of the operational Research Society*, 61: 1619-1631.

[81] Salhi, S. and Gamal, MDH., 2003, "A genetic algorithm based approach for the uncapacitated continuous location-allocation problem", *Annals of Operations Research* 123: 203-222.

[82] Salhi, S., and Sari, M., 1997, "A multi-level composite heuristic for the multi-depot vehicle fleet mix problem", *European Journal of Operational Research*, 103: 95-112.

[83] Shamos, M. I. and Hoey, D., 1975, "Closest point problems", *16th Annual Institute of Electrical and Electronic Engineers Symposium on the Foundations of Computers Science*, 151-162.

[84] Shaw, P., 1998, "Using constraint programming and local search methods to solve vehicle routing problems", In: *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming – Lecture Notes in Computer Science*, 1520: 417-431.

[85] Suzuki, A. and Drezner, Z., 1996, "The *p*-center location problem in an area", *Location Science*, 4: 69-82.

[86] Suzuki, A. and Okabe, A., Using Voronoi diagrams. *Facility Location: A Survey of Applications and Methods* (Drezner, Z., ed.). Springer, New York, 1995, pp. 103-118.

[87] Sylvester, J. J., 1857, "A question in the geometry of situation", *Quarterly Journal of Pure and Applied Mathematics*, 1:79-80.

[88] Sylvester, J. J., 1860, "On Poncelet's approximate valuation of surd forms", *Philos Mag*, 20: 203-222 (Fourth Series).

[89] Teitz, M. and Bart, P., 1968, "Heuristic Methods for Estimating the General Vertex Median of a Weighted Graph", *Operations Research*, 16: 955-961.

[90] Valinsky, D. A., 1955, "A Determination of the Optimum Location of Fire-Fighting Units in New York City ", *Operations Research*, 3: 494-512.

[91] Vijay, J., 1985, "An algorithm for the *p*-center problem in the plane", *Transportation Science*, 19: 235-245.

[92] Watson-Gandy, C.D.T, 1984, "The multi-facility min--max Weber problem", *European Journal of Operational Research*, 18: 44-50.

[93] Wei, H., Murray, A. T. and Xiao, N., 2006, "Solving the Continuous Space p-Centre Problem: Planning application issues", *IMA Journal Management Mathematics*, 17: 413-425.

[94] Wesolowsky, G. O., 1972, "Rectangular Distance Location under the Minimax Optimality Criterion", *Transportation Science*, 6: 103-113.

[95] Young, H. A., 1963, "On the Optimum Location of Checking Stations", *Operations Research*, 11: 721-731.

[96] Zainuddin, Z. M. and Salhi, S., 2007, "A Perturbation-Based Heuristic for the Capacitated Multisource Weber Problem", *European Journal of Operational Research*, 179: 1194-1207.

Appendices

Appendix *A*: Some mathematical results

Appendix *A1*: (Thales' theorem)

Any point on the circumference with the two points on the two ends of a diameter makes right triangle.

This depends on the following facts:

- the sum of the angles in a triangle is equal to two right angles (180°),
- the base angles of an isosceles triangle are equal.

Let O be the centre of the circle.

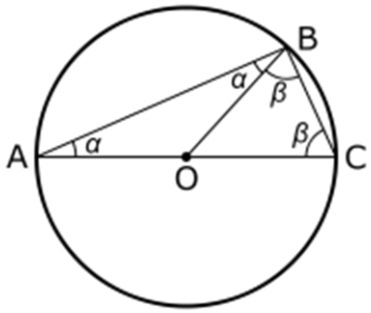


Figure *A1*: illustrate a right-angled triangle at B

Since OA = OB = OC, OBA and OBC are isosceles triangles, and by the equality of the base angles of an isosceles triangle,

$O\hat{B}C = O\hat{C}B$

and $B\hat{A}O = A\hat{B}O$.

Let $\alpha = B\hat{A}O$ and $\beta = O\hat{B}C$.

The 3 internal angles of the ABC triangle are α, α + β and β.

Since the sum of the angles of a triangle is equal to two right angles,

we have $\alpha + (\alpha + \beta) + \beta = 180°$, then

$2\alpha + 2\beta = 180°$

or simply

$\alpha + \beta = 90°$

Appendix *A2*:

There is no intersection between two circles only.

There can be intersection between all the three circles or there would be no intersection at all.

Proof:

In Figure *A2* (1), at any point (*x*) on the perimeter of the circle $L(P_s, P_t)$, the distance from point (*x*) to point P_s multiplied by weight (w_s) is equal to the distance from point (*x*) to point P_t multiplied by weight (w_t).

$\because\ w_s\ d(x, P_s) = w_t\ d(x, P_t)$ (*Feasible solution between two points*)

$\because\ w_s\ d(a, P_s) = w_t\ d(a, P_t)$ (*Optimal solution between two points*)

$\therefore\ w_s\ d(a, P_s) < w_s\ d(x, P_s)$

In Figure *A2* (1), this can also be applied at point (*b*) for circle $L(P_s, P_u)$ and at point (*c*) for the circle $L(P_t, P_u)$.

$w_t\ d(n,\ P_t) = w_u\ d(n,\ P_u) = w_s\ d(n\ , P_s)$, because this point (*n*) is the intersection point of all these three circles (optimal solution between three points).

$w_t\ d(m,\ P_t) = w_u\ d(m,\ P_u) = w_s\ d(m,\ P_s)$, because this point (*m*) is the intersection point of all these three circles (feasible solution between three points).

a) If there is intersection between the circles (at point (n) and (m)):

$w_s\ d(n,\ P_s) = w_t\ d(n,\ P_t) = w_u\ d(n,\ P_u)$. Because the point (*n*) is located on the perimeters of the three circles at the same time (all the three circles intersect at the point (*n*)). This can be also applied at point (*m*).

From circle $L(P_s, P_t)$, we conclude that $w_s\ d(n,\ P_s) = w_t\ d(n,\ P_t) \rightarrow (1)$

And from circle $L(P_s, P_u)$, we conclude that $w_s\ d(n,\ P_s) = w_u\ d(n,\ P_u) \rightarrow (2)$

From (1) and (2) we conclude that $w_t\ d(n,\ P_t) = w_u\ d(n,\ P_u)$, and this relationship is determined by the third circle $L(P_t, P_u)$.

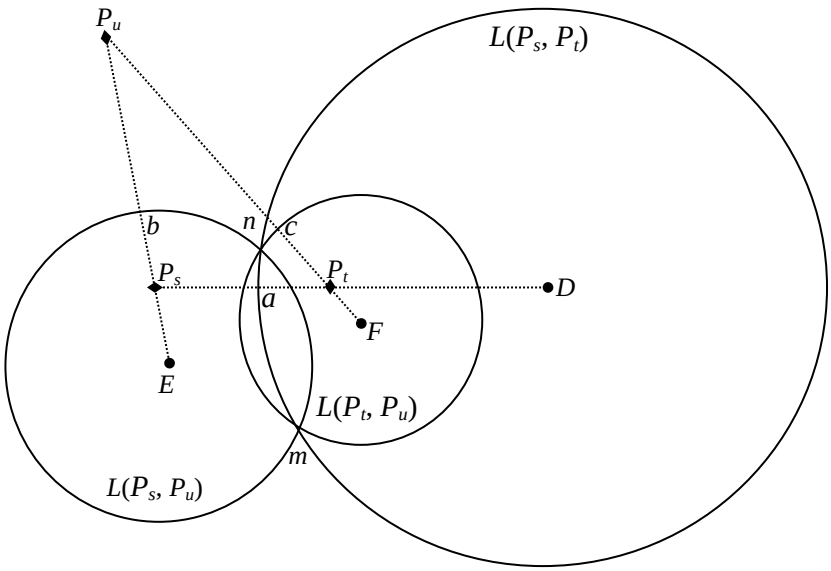


Figure *A2* (1): Case of an intersection between circle

b) If there is no intersection between the circles:

However, if there is no intersection between the circles (there is no point located on the perimeters of the two circles at the same time). This means that any point (x) is located on the perimeter of the first circle $L(P_t, P_u)$ or on the second circle $L(P_s, P_u)$ will lead to:

$w_s\ d(x,\ P_s) \neq w_t\ d(x,\ P_t)$ and $w_s\ d(x,\ P_s) \neq w_u\ d(x,\ P_u)$, which lead to $w_t\ d(x,\ P_t) \neq w_u\ d(x,\ P_u)$

From the above we can conclude that all these three circles are meeting at some point or do not meet at all, namely, that cannot be there a cross between only two circles.

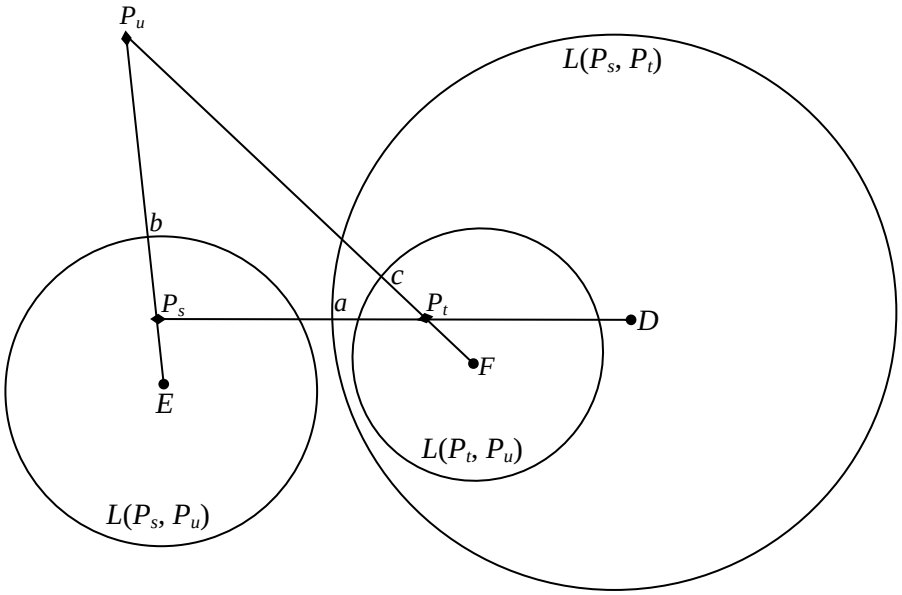


Figure *A2* (2): Case of no intersection between circles

Appendix *A3*:

Situation of a circle with other circles

Generally, the situation of a circle with other circle can be divided into five cases. In order to be able to clarify those cases, we use the following notation:

$r_a =$ the radius of circle (a)

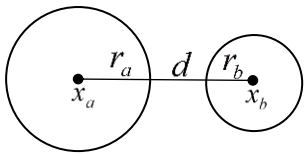
$r_b =$ the radius of circle (b)

$x_a =$ the centre of circle (a)

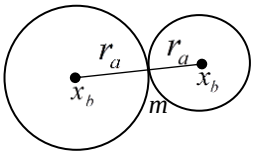
$x_b =$ the centre of circle (b)

d = the distance between the centre of the circle (a) and centre of the circle (b).

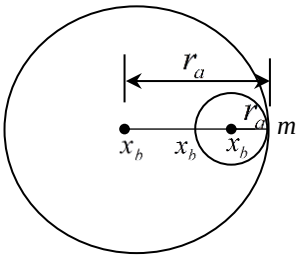
1) if $d > r_a + r_b$ the circle (a) ∩ circle (b) = ∅ (there is no intersection between d > circle (a) and circle (b)).



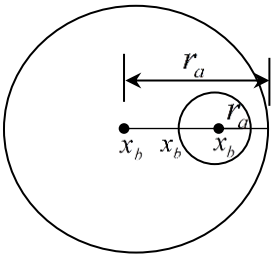
2) if $d = r_a + r_b$ the circle (a) ∩ circle (b) ={m} (there is one intersection in m), the circle (a) touches the outside of the circle (b) but does not cross it.



3) if $d = r_a - r_b$ the circle (a) ∩ circle (b) ={m} (there is one intersection which is in m), the circle (b) touches the inside of the circle (a) but does not cross it.



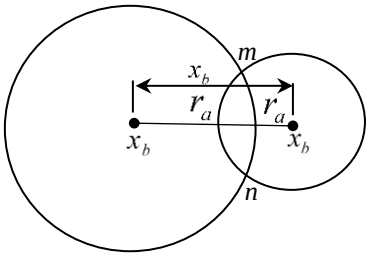
4) if $d < r_a - r_b$ the circle (a) ∩ circle (b) = ∅ (there is no intersection), the circle (b) is located inside the circle (a).



5) if $r_a - r_b < d < r_a + r_b$

where $r_a \geq r_b$

the circle (a) ∩ circle (b) ={m, n} (there is two intersections which are m and n).



Appendix *A4*:

If two circles are crossed, the straight line that connecting between the centre of the first circle and the centre of the second circle is perpendicular to the chord (the straight line that connecting between the intersection points of the two circles), and halves it.

We use this symbol (≅) to indicate congruence.

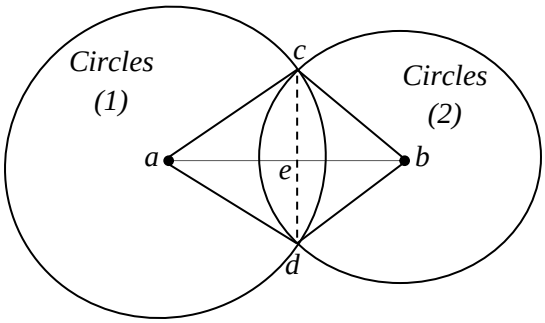


Figure *A5*: intersections between circles

a) The Line segment that connecting between centres of the circles is perpendicular to the chord.

We know that if the sides of the triangle are equal to the sides of the other triangle they would be congruent. In Figure 5, the triangle *acb* and the triangle *adb* would be congruent. Because:

1) Line segment $\overline{ab}$ is one of the sides of the triangle *acb* and the triangle *adb* at the same time.
2) Line segment $\overline{ac}$ is $\cong$ to line segment $\overline{ad}$ (they are the radius of the first circle). $\rightarrow$ (1).
3) Line segment $\overline{bc}$ is $\cong$to line segment $\overline{bd}$ (they are the radius of the second circle).

$\therefore$ The triangles *acb* and *adb* would be congruent.

Hence we conclude that the angle $cab \cong dab \rightarrow (2)$.

$\therefore$ The triangle *acb* is an isosceles triangle (two sides are the same length) $\rightarrow$ (from (1)).

The line segment $\overline{ae}$ halves the angle *cad* $\rightarrow$ (from (2)).

Therefore, from (1), (2) and characteristic of an isosceles triangle, we conclude that the line segment $\overline{ae}$ is perpendicular to $\overline{cd}$.

b) The straight line that joins centres of the circles halve the chord.

We know that if the two sides of the triangle and the angle (which is between them) are equal to two sides and angle of the other triangle they would be congruent. Therefore, the triangle *ace* and the triangle *aed* would be congruent. Because:

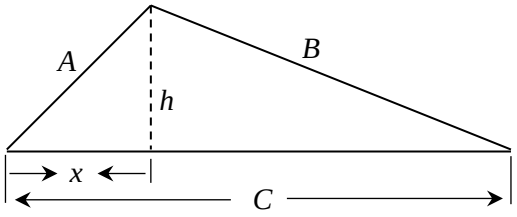
4) Line segment $\overline{ae}$ is one of the sides of the triangle *cae* and the triangle *dae* at the same time.
5) Line segment $\overline{ac}$ is $\cong$ to line segment $\overline{ad}$ (from (1)).
6) Angle *cae* is $\cong$ to angle *dae* (from (2)).

$\because$ The triangles cae and dae would be congruent.

$\therefore$ Line segment $\overline{ce}$ is $\cong$ to line segment $\overline{de}$.

Hence we conclude that the line segment $\overline{ab}$ halves the line segment $\overline{cd}$.

c) Proof for $x = \dfrac{A^2 + C^2 - B^2}{2C}$



$\because x^2 + h^2 = A^2 \implies h^2 = A^2 - x^2$

$\because h^2 = B^2 - (C - x)^2 \implies h^2 = B^2 - C^2 - x^2 + 2xC$

$\therefore A^2 - x^2 = B^2 - C^2 - x^2 + 2xC$

$A^2 = B^2 - C^2 + 2xC$

$2xC = A^2 + C^2 - B^2$

$$x = \frac{A^2 + C^2 - B^2}{2C}$$

Appendix *B*: Results of the *p*-centre

Appendix *B*1: Tables for the unweighted case

n=500 *p*=3	The Original algorithm (50 iteration)			*V4*		
	Z	CPU Time	Number best	*Z*	Total iteration	Number best
1	47.566	1.01	1	47.566	100	2
2	48.797	1.34	1	48.797	67	1
3	48.169	1.13	3	48.169	93	6
4	47.877	1.66	11	47.877	89	18
5	46.632	1.19	12	46.632	98	19
6	46.971	1.14	2	46.971	97	2
7	45.621	1.44	6	45.621	91	10
8	48.283	1.34	2	48.283	77	4
9	47.416	1.08	2	47.416	84	3
10	48.665	1.11	3	48.665	90	3
Average	47.600	1.24	4.30	47.600	88.60	6.80

Table B1 (1): The solution for the unweighted case when n= 500 and p=3

n=500 *p*=5	The Original algorithm (50 iteration)			*V4*		
	Z	CPU Time	Number best	*Z*	Total iteration	Number best
1	30.071	1.51	1	30.071	91	1
2	30	1.59	1	30	93	3
3	30.088	2.04	1	30.088	93	2
4	30.364	3.18	3	30.364	98	5
5	29.778	1.90	1	29.778	90	2
6	29.567	1.90	1	29.567	99	1
7	30.177	1.51	0	29.925	97	1
8	30.225	1.74	1	30.225	92	1
9	30.058	1.88	1	30.058	85	1
10	29.981	1.43	1	29.981	90	1
Average	30.031	1.87	1.10	30.006	92.80	1.80

Table B1 (2): The solution for the unweighted case when n= 500 and p=5

n=500 *p*=10	The Original algorithm (50 iteration)			*V4*		
	Z	CPU Time	Number best	*Z*	Total iteration	Number best
1	21	3.23	0	20.059	81	1
2	20.797	3.47	0	20.607	83	1
3	21.006	4.53	1	20.555	87	1
4	21.030	2.38	1	20.402	85	1
5	20.749	3.40	1	20.749	85	1
6	20.887	3.23	0	20.881	88	1
7	20.452	2.25	1	20.452	88	1
8	21.360	4.31	0	20.555	91	1
9	20.590	2.29	0	20.346	90	1
10	20.524	3.18	1	20.524	87	1
Average	20.839	3.23	0.50	20.513	86.50	1

Table B1 (3): The solution for the unweighted case when n= 500 and p=10

n=500 *p*=15	The Original algorithm (50 iteration)			*V4*		
	Z	CPU Time	Number best	*Z*	Total iteration	Number best
1	17.103	4.04	0	17.054	61	1
2	17.263	3.97	1	17.263	57	1
3	17.529	4.10	0	17.095	62	1
4	17.443	3.91	1	17.443	60	1
5	17.204	4.13	1	17.204	63	1
6	17.321	4.07	1	17.321	64	1
7	17.335	4	1	17.335	62	1
8	16.819	4.25	1	16.819	61	1
9	17.616	4.04	0	17.267	63	1
10	17.069	4.11	1	17.069	61	1
Average	17.270	4.06	0.70	17.187	61.40	1

Table B1 (4): The solution for the unweighted case when n= 500 and p=15

n=500 *p*=20	The Original algorithm (50 iteration)			*V4*		
	Z	CPU Time	Number best	*Z*	Total iteration	Number best
1	14.534	5.17	1	14.534	62	1
2	15	4.77	0	14.976	61	1
3	14.857	5.11	1	14.857	61	1
4	14.705	4.71	1	14.705	62	1
5	15.008	4.87	1	15.008	59	1
6	14.385	4.89	1	14.385	57	1
7	14.840	5.17	1	14.840	64	1
8	14.889	4.69	1	14.889	62	1
9	15.008	5.96	0	14.835	60	1
10	15.182	4.81	0	14.935	60	1
Average	14.841	5.02	0.70	14.796	60.80	1

Table B1 (5): The solution for the unweighted case when n= 500 and p=20

Appendix *B2*: Tables for the weighted case

n=500 *p*=3	The Original algorithm (50 iteration)			*W4*		
	Z	CPU Time	Number best	*Z*	Total iteration	Number best
1	378.430	1.26	2	378.430	93	3
2	383.259	1.25	2	383.259	92	4
3	417.272	1.12	9	417.272	85	18
4	406.263	0.98	4	406.263	104	8
5	384.066	1.25	4	384.066	89	6
6	432.753	1.20	0	429.941	88	2
7	404.780	1.50	12	404.780	71	22
8	388.488	1.28	3	388.488	103	9
9	430.844	1.09	3	430.844	94	9
10	436.021	1.17	0	434.626	89	1
Average	406.218	1.21	3.90	405.797	90.80	8.20

Table B2 (1): The solution for the weighted case when n= 500 and p=3

n=500 *p*=5	The Original algorithm (50 iteration)			*W4*		
	Z	CPU Time	Number best	*Z*	Total iteration	Number best
1	262.450	2.06	1	262.450	91	1
2	253.982	1.90	2	253.982	89	2
3	255.964	1.81	1	255.964	84	1
4	257.519	1.61	2	257.519	85	2
5	242.913	2.14	1	242.912	82	2
6	255.997	1.92	2	255.997	77	2
7	270.636	1.67	2	270.636	87	3
8	253.147	1.90	2	253.147	82	3
9	265.047	1.94	2	265.047	89	3
10	270.185	1.84	4	270.185	89	8
Average	258.784	1.88	1.90	258.784	85.50	2.70

Table B2 (2): The solution for the weighted case when n= 500 and p=5

n=500 *p*=10	The Original algorithm (50 iteration)			*W4*		
	Z	CPU Time	Number best	*Z*	Total iteration	Number best
1	178.401	3.34	0	176.776	78	1
2	174.642	3.04	1	174.642	77	1
3	184.247	2.81	1	184.247	76	1
4	177.475	2.65	1	177.475	83	1
5	170.282	2.90	1	170.282	83	2
6	180.056	2.96	0	173.490	81	1
7	187.300	2.48	0	186.395	78	1
8	176.967	2.81	1	176.967	79	1
9	182.002	2.79	1	182.002	88	1
10	178.437	2.54	1	178.437	80	1
Average	178.981	2.83	0.70	178.071	80.30	1.10

Table B2 (3): The solution for the weighted case when n= 500 and p=10

n=500 *p*=15	The Original algorithm (50 iteration)			*W4*		
	Z	CPU Time	Number best	*Z*	Total iteration	Number best
1	140.978	3.34	1	140.978	65	1
2	148.664	3.28	0	140.151	67	1
3	141.226	3.10	1	141.226	67	1
4	149.071	3.09	1	149.071	73	2
5	139.717	2.96	0	136.713	72	1
6	148.664	3.12	1	148.664	68	1
7	145.922	3.06	1	145.922	69	1
8	147.960	3.32	1	147.960	68	1
9	147.902	3.20	1	147.902	67	2
10	150.579	2.96	1	150.579	70	1
Average	146.068	3.14	0.80	144.917	68.60	1.20

Table B2 (4): The solution for the weighted case when n= 500 and p=15

n=500 *p*=20	The Original algorithm (50 iteration)			*W4*		
	Z	CPU Time	Number best	*Z*	Total iteration	Number best
1	128.871	3.68	0	121.147	64	1
2	122.983	3.28	1	122.983	63	1
3	127.269	3.67	1	127.269	63	2
4	133.216	3.78	1	133.216	60	1
5	116.464	3.43	1	116.464	63	1
6	124.482	3.46	0	119.552	67	1
7	123.568	3.50	1	123.568	65	1
8	121.330	3.82	1	121.330	62	1
9	135.477	3.54	0	128.252	67	1
10	126.402	3.73	1	126.402	62	1
Average	126.006	3.59	0.70	124.018	63.60	1.10

Table B2 (5): The solution for the weighted case when n= 500 and p=20

Appendix *C*: Results for the Multi-Start

Appendix *C1*:

Table *C1*: Best iteration number for the Multi-Start

n / *p*	439	575	783	1002	1323
10	2647	6477	7579	501	7299
20	8579	105	2235	9315	5110
30	214	1254	987	6462	3964
40	8122	9218	9572	8270	3511
50	2321	6629	3534	8775	2126
60	6560	9600	6718	1555	5260
70	321	4536	2210	7803	4928
80	286	3859	5401	3149	3724
90	6803	2341	4386	5964	3724
100	7265	1674	5226	102	4512
Average	4311.80	4569.30	4784.80	5189.60	4415.80
Max	8579	**9600**	9572	9315	7299

Appendix *C2*:

Table *C2*: CPU time of the Multi-Start algorithm (for *p*=10 to 100 in increment of 10), Deviation (%) of CPU time for VNS2 (FNV4) (with and without learning) and VNS (CNV3)

n	*p*	Total CPU time (10,000 iterations)	Deviation (%)				
			VNS (CNV3)	VNS2 (*FNV4*)		Chen and Chen's results (Continuous Solutions)	
				No Learning	With Learning	Improved relaxation (*k*=7)	Binary relaxation (*k*=6)
439	10	435.06	-97.12	-97.90	-98.71	-93.48	-99.39
	20	751.33	-98.93	-98.79	-95.89	-96.21	-99.44
	30	1018.99	-98.57	-98.06	-98.17	-86.74	-99.18
	40	1171.13	-96.81	-97.71	-98.52	-83.34	-97.07
	50	1730.06	-90.18	-18.80	-77.69	-86.22	-97.34
	60	1984.20	-93.79	-85.65	-65.82	-88.13	-99.12
	70	2087.36	-98.26	-43.11	-29.74	-89.53	-99.23
	80	1943.06	-66.12	-60.36	-67.57	-87.83	-99.07
	90	1988.14	-55.76	-52.08	-58.32	-89.66	-98.83
	100	1866.30	-21.71	-79.65	-56.01	-88.18	-98.90
	Average	1497.56	-81.73	-73.21	-74.64	-88.93	-98.76
575	10	541.70	-97.42	-83.74	-59.64	N/A	N/A
	20	943.64	-76.58	-95.72	-88.07	N/A	N/A
	30	1190.15	-79.64	-54.51	-0.302	N/A	N/A
	40	1436.05	-60.13	-55.56	-76.23	N/A	N/A
	50	1766.35	-21.71	-75.04	-49.31	N/A	N/A
	60	1789.30	-89.54	-20.20	-49.45	N/A	N/A
	70	2143.63	-33.05	-24.16	-8.64	N/A	N/A
	80	2167.66	-14.64	-48.47	-4.31	N/A	N/A
	90	2307.93	-40.44	-16.08	-26.76	N/A	N/A
	100	2531.67	-45.85	-1.71	-6.35	N/A	N/A
	Average	1681.81	-55.90	-47.52	-36.91	N/A	N/A
783	10	909.64	-96.28	-54.97	-98.76	N/A	N/A
	20	1555.44	-53.55	-86.95	-96.41	N/A	N/A
	30	2055.59	-75	-87.70	-44.81	N/A	N/A
	40	2403.09	-39.61	-21.90	-22.59	N/A	N/A
	50	2514.47	-76.16	-20.60	-71.57	N/A	N/A
	60	2842.81	-5.07	-36.35	-10.40	N/A	N/A
	70	3154.78	-6.07	-5.27	-49.62	N/A	N/A
	80	4466.13	-2.57	-25.35	-36.96	N/A	N/A
	90	3646.99	-7.75	-56.64	-20.81	N/A	N/A
	100	4075.51	-36.41	-0.82	-36.49	N/A	N/A
	Average	2762.45	-39.85	-39.66	-48.84	N/A	N/A
1002	10	947.82	-97.50	-96.55	-95.41	N/A	N/A
	20	1627.17	-95.08	-37.01	-89.16	N/A	N/A
	30	2562.06	-65.53	-93.91	-94.75	N/A	N/A
	40	2959.21	-71.90	-91.43	-69.57	N/A	N/A
	50	3682.88	-27.86	-73.94	-8.02	N/A	N/A
	60	4520.70	-35.30	-22.88	-54.82	N/A	N/A
	70	6640.19	-1.34	-12.72	-32	N/A	N/A
	80	7026.19	-9.43	-37.66	-73.85	N/A	N/A
	90	6883.15	-19.41	-43.27	-3.09	N/A	N/A
	100	7131.51	-30.99	-83.40	-59.14	N/A	N/A
	Average	4398.09	-45.43	-59.28	-57.98	N/A	N/A
1323	10	1584.92	-97.24	-92.38	-93.71	N/A	N/A
	20	2439.18	-79.21	-54.79	-83.56	N/A	N/A
	30	3454.57	-57.81	-64.61	-4.88	N/A	N/A
	40	4093.15	-38	-52.77	-43.21	N/A	N/A
	50	5677	-74.06	-8.43	-38.76	N/A	N/A
	60	6527.83	-30.60	-40.33	-23.18	N/A	N/A
	70	7515.28	-2.27	-12.72	-29.97	N/A	N/A
	80	7905.08	-37.57	-2.68	-7.10	N/A	N/A
	90	8417.76	-18.16	-4.46	-70.28	N/A	N/A
	100	9015.06	-72.97	-3.49	-89.70	N/A	N/A
	Average	5662.98	-50.79	-33.67	-48.44	N/A	N/A
Overall	Average	3200.58	-54.74	-50.67	-53.36	N/A	N/A

k:is the best recorded value in Chen and Chen (2009).

Appendix *D*: Results with version β in forbidden regions

The following Table shows the results of Enh 2 with forbidden regions for the three selection rules (*a*, *b*, *c*), when $\beta = 0.1$.

Table *D* : Effect of forbidden region on the three selection rules (*a*, *b*, *c*) of Enh2 with $\beta = 0.1$

n = 439 TSP-Lib	The optimal solution Z	Selection (*a*)				Selection (*b*)				Selection (*c*)			
p		Deviation % (Z)	# Switch-es	# Cplex Calls	CPU Time (sec)	Deviation % (Z)	# Switch-es	# Cplex Calls	CPU Time (sec)	Deviation % (Z)	# Switch-es	# Cplex Calls	CPU Time (sec)
10	1716.5099	5.95	4	53	13.53	5.95	4	54	14.34	5.95	5	65	17.68
20	1029.7148	0	12	142	29.35	0	9	105	28.06	0	12	135	35.00
30	739.19297	6.48	9	102	22.43	6.75	11	107	26.32	6.48	10	108	29.56
40	580.00539	7.37	12	108	35.47	7.82	14	135	40.30	7.37	16	144	47.71
50	468.54162	12.18	18	167	40.33	8.04	21	184	56.38	10	25	224	55.36
60	400.19527	9.37	23	195	52.16	10.60	24	219	53.59	10.19	19	166	44.16
70	357.94553	11.37	35	292	103.22	9.71	31	273	83.55	9.38	43	339	115.63
80	312.5	18.93	20	167	54.01	11.42	35	287	90.92	18.93	21	174	55.98
90	280.90256	8.64	36	281	111.82	11.25	36	304	117.56	8.64	44	353	117.20
100	256.68019	9.12	36	290	118.57	8.89	26	198	79.11	8.89	32	260	90.66
Average	614.21882	8.94	20.5	179.7	58.09	8.04	21.1	186.6	59.01	8.58	22.7	196.8	60.89

Appendix *E*: Contributions to the Research

The main research contributions are highlighted. These include presentations at international conferences in Operational Research, as well as the papers that are published, under revision or in preparation.

Journals

1- 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: 606–621. This is mainly based on Chapters 2 and 3.

2- Elshaikh A., Salhi S., Brimberg J., Mladenović N., Nagy, G. & Callaghan, B., (2014). *Adaptive Perturbation-Based Heuristics: An Application for the Continuous p-centre Problem*. Computers & Operations Research, (revision submitted). This is mainly based on Chapter 4.

3- Elshaikh A., Salhi S., & Brimberg J., & Mladenović N. (2014). *Reformulation Local Search Methodology for the Continuous p-centre Problem*, (in preparation). This is based on Chapter 5.

Conferences

1- Salhi S., & Brimberg J., Mladenović N., & Elshaikh A. *Variable neighbourhood search for the Continuous p-centre problem*. MIC2011, Udine, Italy, 25-28 Jul 2011. This research is from Chapter 3.

2- Elshaikh A., Salhi S., & Nagy, G. *Variable Neighbourhood Search (VNS) Based Approaches for Continuous p-entre location problems*. XXVI EURO-INFORMS, Rome, 1-4 July 2013. This is mainly based on Chapters 2 and 3.

3- Elshaikh A., Salhi S. & Nagy, G. *A Perturbation-based heuristic with learning for solving p-centre location problem*. 20th conference of the international federation of operational research societies, Barcelona, 13-18 July 2014. This work is from Chapter 4.