The continuous *p*-centre problem: An investigation into variable neighbourhood search with memory

Abdalla Elshaikh, Said Salhi, and Gábor Nagy

Centre for Logistics and Heuristic Optimisation (CLHO), Kent Business School, University of Kent, Canterbury, UK
{ae201, s.salhi, g.nagy}@kent.ac.uk

Abstract

A VNS-based heuristic using both a facility as well as a customer type neighbourhood structure is proposed to solve the *p*-centre problem in the continuous space. Simple but effective enhancements to the original Elzinga-Hearn algorithm as well as a powerful 'locate-allocate' local search used within VNS are proposed. In addition, efficient implementations in both neighbourhood structures are presented. A learning scheme is also embedded into the search to produce a new variant of VNS that uses memory. The effect of incorporating strong intensification within the local search via a VND type structure is also explored with interesting results. Empirical results, based on several existing data set (TSP-Lib) with various values of *p*, show that the proposed VNS implementations outperform both a multi-start heuristic and the discrete-based optimal approach that use the same local search.

Keywords- *p*-centre problem, continuous space, variable neighbourhood search with memory, adaptive search, Elzinga-Hearn algorithm.

1 Introduction

In the *p*-centre problem, the objective is to locate a given number (*p*) of facilities in order to minimize the maximum distance from a set of fixed points to their closest facilities. In this study, we investigate the case where the facilities can be located anywhere in the plane. This is contrary to the commonly used case where the facility locations are restricted to a candidate set of potential sites. The continuous solution though has some weaknesses in terms of practicality can be of help to identifying potential sites that are nearer to the best locations as gathering the data can, in some situations, be expensive. Also, the information obtained can be used as a green field solution for assessing the company's chosen facilities. The most cited application of the *p*-centre problem involves the location of emergency facilities where

response times are critical. Thus, to obtain an 'equitable' solution, the objective is framed as the minimization of the worst response time instead of the average response time. Related economical implications include among others the recent work by Murray and Wei (2013) and the one by Lu (2013). The former used set covering and GIS to obtain the least number of facilities to cover the entire area of study. Two real life applications are used where the first one aims at locating emergency sirens in Dublin (Ohio) whereas the second is about the siting of fire station in Elk Grove (California). The paper by Lu (2013) explores the use of *p*-center as part of emergency management while taking into account uncertain demand as this is very common in emergency logistics systems aiming at responding to natural disasters. A case study using the earthquake in Taiwan in 1999 is adopted.

The existing research on the *p*-centre problem deals mainly with the network (or discrete) formulation of the problem; this version is usually referred to as the vertex *p*-centre problem. For a fixed value of *p*, the vertex *p*-centre problem can be solved in polynomial time. This can be done by evaluating each of the $O(n^p)$ possible combinations of p facility sites in polynomial time. For more details refer to Chen and Chen (2009) and Salhi and Al-Khedhairi (2010) and references therein.

For the continuous case, efficient solution approaches have been proposed for the one-centre problem ($p = 1$) including Elzinga and Hearn (1972) who devised an exact geometrical approach for solving optimally the problem. Enhancements to speed up the search were also introduced by several authors, see Xu *et al.* (2003) and references therein. For p=2, Drezner (1984*a*) designed an interesting exact algorithm where the idea is to enumerate efficiently all the possible disjoint pairs of subsets (i.e., *n(n-1)/2* possibilities) by using the optimal algorithm for p=1 for each subset. For large values of p ($p \geq 3$), the problem is known to be NP hard (see Megiddo and Supowit, 1984).

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

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

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 fixed point i $(i = 1, \ldots, n)$

$w_i > 0$: the weight of fixed point i $(i = 1, \ldots, n)$

$X = (X_1, ..., X_p)$: 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 $(i = 1, ..., n; j = 1, ..., p)$

The above multiple facility location problem has been examined by a small number of authors, see Plastria (2002) and the references therein. For larger values of *p* and *n*, heuristic methods were developed by Drezner (1984b) and Eiselt and Charlesworth (1986) where the iterative procedures are based on the idea of 'locate-allocate' with the use of the add/drop/swap moves. A Voronoi diagram-based heuristic, that has the flavour of Cooper's well-known locate-allocate strategy, was also proposed in Suzuki and Okabe (1995). This method was generalized by Wei *et al.* (2006) to account for irregular shapes and constraints on the possible locations of the new facilities. Relaxation methods, based on solving optimally small subsets of the original problem, which are then gradually increased in size by a given number of demand points, were developed by Chen and Chen (2009) for both the discrete and the continuous *p*-centre problem with excellent results. This method is interesting as it could generate optimal solutions though it is sensitive to the number of added points. Observing the literature two issues arise. Firstly the optimal methods can solve instances with limited sizes and secondly there are no meta-heuristics only greedy and simple improvement type methods.

The contributions of the present paper include

(i) Enhancements on the Elzinga-Hearn's method which is part of the local search used for the *p*-center problem

(ii) The design of a powerful meta-heuristic namely a VNS by introducing efficient neighbourhood structures and effective enhancements in its local search to solve large instances

(iii) The incorporation of memory in VNS to provide flexibility and guidance to the search

(iv) Extensive computational experiments for large instances leading to new results that are useful for benchmarking purposes including the optimal solutions for the discrete case.

The paper is organised as follows: Enhancements for the Elzinga-Hearn algorithm (i.e. $p = 1$) are described in section 2 alongside an initial application and adaption to the *p*-centre

problem. In section 3, a VNS implementation is produced followed by improvement schemes in the generation of an effective neighbourhood structure in section 4 and enhancements on the local search in section 5. A learning mechanism that systematically responds to the characteristics of a given instance making VNS not memoryless is provided in section 6. In section 7 computational experiments are presented followed by our conclusion and suggestions in the last section.

2 Enhancements to the Elzinga-Hearn algorithm

Though the algorithm is polynomial of the order $O(n^2)$ and hence very fast, any enhancement would seem not to be worthwhile if the aim was to solve the 1-centre problem only. However, our aim is to solve the *p*-centre problem instead where we need to resolve to solving the *1*-center problem a large number of times and therefore the cumulative computational saving would be, in our view, worth considering. For completeness, a brief recall of the Elzinga-Hearn algorithm is first given followed by our proposed enhancements.

2.1 A brief on the Elzinga-Hearn algorithm

The optimal solution for the *1*-centre problem (X) can be obtained with a geometrical-based approach using the following two results.

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

The optimal solution X lies at the intersection of the set $L(P_s, P_t) = \{X : w_s d(P_s, X) = w_t d(P_t, X) \ \ \forall s \neq t = 1,...,n\}$ and the line between the points P_s and P_t.

Let $r = \frac{w_s}{w_t}$. If $r = 1$ the set L reduces to the perpendicular bisector, otherwise L is a circle with radius $\frac{r}{|1-r^2|} d(P_s, P_t)$ and centre $\frac{P_s - r^2 P_t}{1-r^2}$.

Result 2 (*case of 3 critical points, say* P_s, P_t and P_u)

The optimal solution is determined by one of the pairs of points P_s and P_t , or P_s and P_u , or P_t and P_u leading to the points a, b or c respectively, or by all three points in which the solution lies at the intersection of $L(P_s, P_t), L(P_s, P_u)$ and $L(P_t, P_u)$ leading to the points e_1 and e_2.

From these five points $\{a,b,c,e_1,e_2\}$, the choice will reduce to choosing one point either from $\{e_1,e_2\}$ or $\{a,b,c\}$.

In brief, the optimal solution can be determined by one, two or three fixed points only which are referred to, in the literature, as the critical points. Using these interesting results, Elzinga and Hearn (1972) developed the following algorithm (see Figure 1) to find the optimal location for the *1*-centre problem in the continuous space.

Step 1: Choose any two points P_s and P_t. Solve the weighted minimax location problem with P_s and P_t to find X using Result 1and let $Z = w_s d(P_s, X)$

Step 2: If $w_i d(P_i, X) \le Z \ \forall P_i; i = 1,...,n$ stop, else select a point P_u such that $w_u d(P_u, X) > Z$ (i.e., uncovered points).

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

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

Step 5: X is determined by three points. If $w_i d(P_i, X) \le Z \ \forall P_i; i = 1,...,n$ stop; Otherwise choose point P_v such that $w_v d(P_v, X) > Z$ (i.e., uncovered points).

Step 6: - Using P_s, P_t, P_u and P_v select all combinations of two points to find the optimal location X using Result 1, and choose all combinations of three points to find the optimal location X using Result 2.

- Among these solutions, choose X with the largest Z value.
- If the solution is determined by two points, let P_s and P_t be these 2 points and go to *Step* 2; Otherwise (i.e., the solution is found by three points), let the three points be P_s, P_t and P_u and go to *Step* 5.

Figure1: The original Elzinga-Hearn algorithm

Regarding the addition of the 4th point (*Step* 6 of Figure 1), six combinations need to be evaluated only (three using two points and the other three requiring three points). Moreover, if the problem is unweighted, there is no need to check all the six cases.

2.2 The proposed enhancements

Elzinga and Hearn (1972) noted the following weaknesses of their algorithm: (i) selection of the starting points (*Step* 1 of Figure 1) and (ii) the selection of the uncovered points in *Step* 2 and *Step 5* of Figure 1. Attempts to address these shortcomings were made by Hearn and

Vijay (1982), but with less convincing results. Here we propose two simple but effective enhancements for both the weighted and the unweighted cases. The steps of these two enhancements are similar to the original algorithm, except that *Steps* 1, 2 and 5 of Figure 1 are replaced as follows:

Enhancement 1 (Enh 1)

Only *Step* 1 is changed as follows.

Step 1:

- Determine the four corners of the rectangle with horizontal and vertical sides that covers all demand points, namely let $B=\{i_1,i_2,i_3,i_4\}$ with $i_1 = Arg \underset{i=1,...,n}{Min}(w_i x_i)$, $i_2 = Arg \underset{i=1,...,n}{Max}(w_i x_i)$, $j_1 = Arg \underset{j=1,...,n}{Min}(w_i y_i)$ and $j_2 = Arg \underset{j=1,...,n}{Max}(w_i y_i)$.

- Solve the weighted minimax location problem using Result 1 with

$(P_s,P_t) = Arg \underset{i,j\in B}{Max} \frac{w_i w_j}{w_i + w_j} d(P_i,P_j)$ to obtain X and its cost $Z = w_s d(P_s,X)$.

Enhancement 2 (Enh 2)

This is an extension of Enh 1 where the uncovered point is chosen as the one with the greatest weighted distance in *Steps* 2 and 5.

Step 1: Same *Step* 1 as in Enh 1.

Step 2 (choice of P_u) & *Step* 5 (choice of P_v) :

If $w_i d(P_i,X) \le Z \ \forall P_i; i=1,...,n$ stop,

else select a point P_u (or P_v) such that P_u(or P_v) $= Arg \underset{i=1,...,n}{Max}(w_i d(P_i,X) > Z)$ (i.e., the uncovered point that has the greatest weighted distance from the previous solution) .

2.3 Computational experiments

The two enhancements were tested on random instances varying in size from *n*=10 to 100 in increment of 10. For each value of *n*, 100 random instances were tested and average results are reported. The fixed points are randomly generated in a square $(0,100)^2$.

Our two proposed enhancements are found to yield extremely better results than the original implementation. Table 1 provides summary results with the % deviation defined as

$Deviation(\%) = 100\dfrac{CPU_E - CPU_{Orig}}{CPU_{Orig}}$ with CPU_E and CPU_{Orig} refer to the CPU times for Enh 1 (or Enh 2) and the original algorithm respectively.

According to Table 1, both enhancements require fewer iterations than the original algorithm in all cases. The average total number of iterations is 4.46 (1.3+ 3.16) and 2.7 (1.03+1.67) for Enh 1 and Enh 2 respectively compared to 9.61 (3.84+5.77) for the original algorithm. In general, Enh 1 and Enh 2 yield similar time reduction of the original algorithm though Enh 2 is slightly faster on average (58% vs 54%) but slightly slower when $n \geq 70$, see Figure 2.

Table 1: Average CPU time (in second), deviation (%) from the original and the number of iterations (over 100 instances of the unweighted case, from *n* = 10 to 100)

n	Original algorithm			Enhancement 1 (Enh 1)				Enhancement 2 (Enh 2)			
	Average CPU	# Iterations using		CPU Time		# Iterations using		CPU Time		# Iterations using	
		2 points	3 points	Average CPU	Deviation (%)	2 points	3 points	Average CPU	Deviation (%)	2 points	3 points
10	0.12422	2.57	2.80	0.05195	58.1790	0.68	1.37	0.03318	73.2893	0.68	1.12
20	0.13852	3.27	4.26	0.07595	45.1704	0.99	1.88	0.05925	57.2264	0.85	1.26
30	0.11842	3.62	5.07	0.08967	24.278	1.13	2.10	0.07768	34.4030	0.93	1.56
40	0.15604	3.68	5.80	0.07889	49.4425	1.21	2.87	0.06457	58.6196	0.99	1.62
50	0.19849	4.03	6.30	0.11582	41.6495	1.11	3.67	0.08659	56.3756	0.94	1.94
60	0.33901	3.76	6.25	0.10882	67.9007	1.51	3.21	0.08558	74.7559	1.18	1.78
70	0.23006	4.41	6.61	0.09825	57.2937	1.48	3.62	0.09799	57.4068	1.08	1.83
80	0.35598	4.14	6.83	0.10765	69.7595	1.53	4.23	0.14997	57.8712	1.15	1.86
90	0.29601	4.20	7.15	0.13211	55.3698	1.64	4.50	0.17219	41.8297	1.19	2.02
100	0.59411	4.76	6.64	0.1793	69.8204	1.75	4.10	0.20585	65.3515	1.30	1.75
Average		3.84	5.77		**53.8863**	1.3	3.16		**57.7129**	**1.03**	**1.67**

In summary, either Enh 1 or Enh 2 can be used instead of the original algorithm when solving the *1*-centre problem as part of the *p*-centre problem but in this study we propose the following rule:

$$\text{If } n_k \leq 70 \text{ use Enh 2, Else use Enh 1} \qquad (1)$$

where $n_k\,(k = 1, \ldots, p)$ represents the number of customers in the k^{th} cluster $(\sum_{k=1}^{p} n_k = n)$.

To validate this claim further, an extensive testing was carried out based on a large sample with *n*=100 to 1000 with a step size of 50. It was observed that the same trend remains valid. For the weighted case, Enh 2 is always found to outperform Enh 1.

This saving in computational effort can have a massive effect within heuristics that perform the 'locate-allocate' principle a large number of times as will be shown in the next subsection where a simple multi-start procedure is used for the *p*-centre problem.

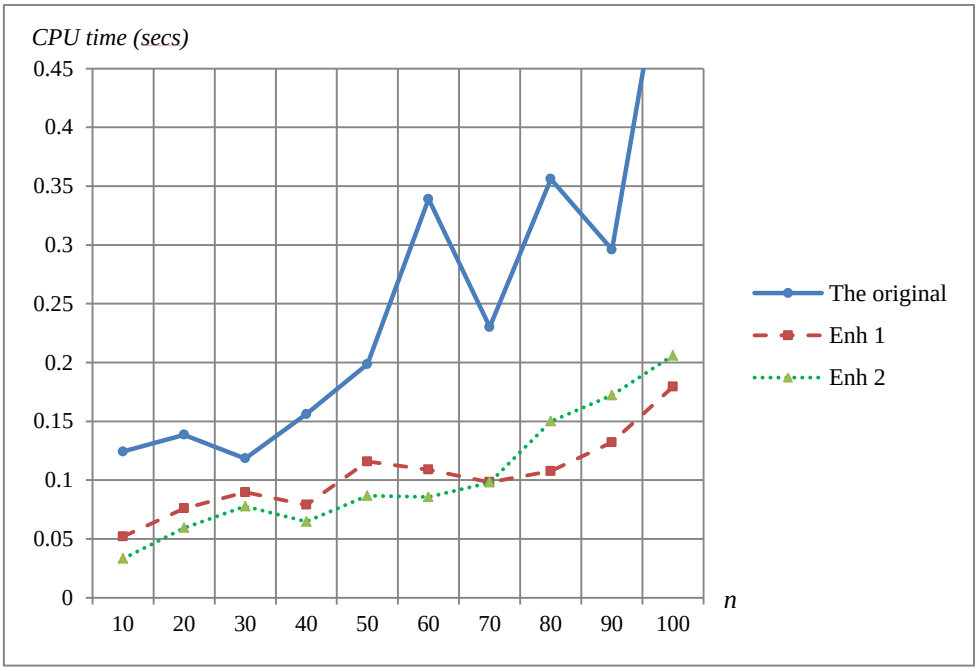

Figure 2: Average CPU time of the original algorithm and the enhancements (Enh 1 and Enh 2) (case of the unweighted problem, n = 10 to 100)

We have also experimented with the following modifications but all combinations of these have proved to be slower than Enh 1 and Enh 2.

(i) Using the farthest two points in terms weighted distance as initial starting points in *Step* 1.

(ii) Using the farthest three points as initial starting points in *Step* 1.

(iii) Selecting the uncovered point in *Steps* 2 and 5 that has the greatest weighted distance from the previous solution.

2.4 Effect of the Enhancements on the planar *p*-centre problem

In this subsection we present computational results of the Multi-Start using the original algorithm (10 random runs) versus those of Enh 1 and Enh 2 for solving the *p*-centre problem. For illustration purposes, we chose one of the TSP-Lib instances (n = 1002) with p varying from 5 to 25 with an increment of 5. We performed 100 iterations for the original Multi-Start and used the required average time as a stopping criterion for the enhanced versions for which we record the number of iterations as well as the corresponding CPU time.

A summary of the comparison between the original algorithm, Enh 1 and Enh 2 is given in Table 2. The results show that there is a significant difference in terms of the total number of iterations between the original algorithm and those of Enh 1 and Enh 2. The average deviations from the original algorithm (100 iterations) for Enh 1 and Enh 2 are 37.20% and 31.60% respectively. In terms of CPU time, the deviations are -28.55% and -25.46%. Note that this saving could be made even larger if the hybrid rule (1) was used instead, as this could have been possible especially when $p = 15$ and 20. This is not performed here purposely as it may disguise the effect of the two enhancements.

Table 2: Results of the Multi-Start using the original algorithm, Enh 1 and Enh 2 (for 100 iterations, n=1002, from p = 5 to 25)

n = 1002 TSP-Lib	The Original algorithm		Enh 1				Enh 2			
P	# iterations	Average CPU Time	CPU Time for 100 iterations	Corresponding # iterations	Improvement Deviations (%)		CPU Time for 100 iterations	Corresponding # iterations	Improvement Deviations (%)	
					CPU Time 100 iterations	Iterations			CPU Time 100 iterations	Iterations
5	100	10.48	5.97	170	43.03	70	7.35	140	29.87	40
10	100	13.56	9.49	141	30.02	41	10.06	132	25.81	32
15	100	18.52	13.78	133	25.59	33	14.67	125	20.79	25
20	100	21.57	17.15	122	20.49	22	16.34	123	24.25	23
25	100	27.22	20.79	120	23.62	20	19.99	138	26.56	38
Average	100			137.20	28.55	**37.20**		131.60	25.46	31.60

3 A VNS-based approach for the *p*-centre problem

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 based on those used for the multi-source Weber problem (continuous *p*-median problem) with some additional changes to cater for the properties of the minimax objective function. These include customer-based moves (e.g., the removal/addition of one or more customers from a region), and facility-based ones (e.g., opening/closing one or more facilities).

3.1 A Basic customer-based VNS

The steps of the VNS that uses a customer-based neighbourhood, and which we call VNS (CN) for short, are given in Figure 3.

Step 0: Specify $K_{\max}$ and $CPU_{\max}$ and set $Time = 0$. Define the neighbourhood structures

$CN_k; k = 1,..., K_{\max}$

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

Step 2: While $k < K_{\max}$ do

- *Step* 2a: Generate a neighbouring solution $X' \in CN_k(X)$ ***"Shaking Part"***
- *Step* 2b: apply the Elzinga-Hearn algorithm or the enhanced versions for the affected clusters. Let X' be the new solution ***"Continuous locations"***
- *Step* 2c: Apply a local search to obtain X'' starting from X' ***"Local Search Part"***
- *Step* 2d: If $Z(X'') < Z(X)$ set $X = X''$ and $k = 1$, else set $k = k + 1$ ***"Evaluation Part"***

Step 3: Record Time.

If $Time > CPU_{\max}$ record the incumbent solution X and stop.
Else set $k = 1$ and go to step 2.

Figure 3: A basic Customer-Based VNS Algorithm (VNS(CN))

Explanation of some of the 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 ($K_{\max} = \lceil \sqrt{p} \rceil$).

Step 1 (the initial solution)

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

Step 2b (the solution of the 1-centre problem)

Our enhancements on the Elzinga-Hearn algorithm using rule (1) are applied to solve the *1*-centre problem for both the source and the destination clusters (i.e., the affected clusters). Note that this step could also be inserted at the beginning of the local search in *Step 2c*.

Step 2c (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 centre $X_j (j = 1,..., p)$, allocate each customer to its nearest centre and define for each centre j, the subset W_j, as

$$W_j = \{(P_i)_{i=1,...,n} : d(P_i, X_j) = \min_{k=1,...,p} d(P_i, X_k)\}, \quad j = 1,..., p$$

(ii) In each subset $W_j (j = 1,..., p)$, determine the optimal location X_j using (1).

(ii) While there is a change in at least one of the subset W_j or 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 5.

Customer-based VNS(CN) enhancement

Given that any circle of minimum radius can be determined by two or three critical points on its circumference or simply by a singleton point, we build our enhancement by taking this information into account. A preliminary study showed that allocating a critical point, instead of a non-critical point, to another facility is likely to be more efficient. In this enhancement, which we call VNS(CN), the critical points of the largest circle are allocated to the other facilities. The only step of Figure 3 which is changed is *Step* 0. This is replaced by

Step 0: Define $CN_k; k = 1,..., K_{\max}$ as the sequence of neighbourhood structures representing the k critical points of the largest circle with $K_{\max}$ being 2 or 3 depending on the number of critical points that define the largest circle at a given iteration.

3.2 The Facility-based VNS

In this section, the facility-based neighbourhood algorithm, VNS(FN) for short, is presented. Its steps are similar to those of the VNS(CN) given in Figure 3 except that in the shaking part, k open facility locations are selected randomly and inserted into other places. These facilities can be located either in the discrete space (fixed points) or in the continuous space. Therefore, this type of neighbourhood which we denote by $FN_k(X); k = 1,..., K_{\max}$ can be classified under two categories namely VNS1(FN) and VNS2(FN), which are defined as follows:

Algorithm VNS1 (FN)

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

$$FN_k(X) = X \setminus \bigcup_{r=1}^{k} X_r \vee \bigcup_{r=1}^{k} X_r^{'} \text{ where } X_r \in X \text{ and } X_r^{'} \in \{P,...,P_n\} - X \qquad (2)$$

The main steps of VNS1(FN) are similar to VNS(CN) of Figure 3 except that *Step 0*, *Step 2a* and *Step 2b* are replaced as follows:

Step 0: Define $FN_k(X); k = 1,...,K_{\max}$ using (2) with $K_{\max} = \left\lceil \sqrt{p} \right\rceil$

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

Step 2b: This step is now void as there is no destination cluster or source cluster.

Algorithm VNS2(FN)

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

$$FN_k(X) = X \setminus \bigcup_{r=1}^{k} X_r \vee \bigcup_{r=1}^{k} X_r^{'} \qquad \text{where } X_r \in X \text{ and } X_r^{'} \in S \subset \Re^2$$

$$\text{with } S = \{(x,y) \in \Re^2 : \underset{i=1,...,n}{Min}(a_i) \le x \le \underset{i=1,...,n}{Max}(a_i) \,\&\, \underset{i=1,...,n}{Min}(b_i) \le y \le \underset{i=1,...,n}{Max}(b_i)\} \qquad (3)$$

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

Step 0: Define $FN_k(X); k = 1,...,K_{\max}$ using (3) with $K_{\max} = \left\lceil \sqrt{p} \right\rceil$

Step 2a: Generate $X' \in FN_k(X)$ using (3)

Step 2b: This step is also void here as there is no destination cluster or source cluster.

Based on a preliminary experiment, the performance of VNS2(FN) is found to be relatively better than VNS1(FN). We therefore concentrate on proposing simple but effective enhancements on VNS2(FN). We first develop an effective neighbourhood structure which is then followed by enhancements on the local search.

4 A New Neighbourhood Structure

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).

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. However, 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 instead the facility locations of the larger circles (1^{st} largest, the 2^{nd} largest,..., the $(K_{\max})^{th}$ largest circle) including the facilities that are around them. This idea can be further refined by focussing on the largest circle and the facilities that are around it only and then locate any facility removed randomly in the largest circle and in its surrounding areas defined by its neighbouring circles which we will explore next. This is achieved using the following two neighbourhood definitions namely the neighbourhood attraction and the neighbourhood removal.

For the sake of simplicity let's index the largest circle as C_1 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 based on their distances from the largest circle using the distance measure $d(X_j, X_1); j = 2,..., p$. The following additional 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 $C_j; j = 1,..., p$

$CP_j = \{P_i \in W_j : d(P_i, X_j) = R_j; i = 1,..., n\}; j = 1,..., p$: the set of critical points of C_j $(|CP_j| \leq 3); j = 1,..., p)$

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

$CR_j = \hat{C}_j \vee \bigcup_{l \in CP_j} RC_{jl}$: 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 regions

$CC_{k'}$: the facilities encompassed by the artificial circle centered 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.

4.1 Facility Attraction

For example, consider Figure 4 which shows three regions (i.e., RC_{1l} with l representing the critical points a_1, a_2 and a_3). It can be shown that these 3 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 p_1, and therefore a_1 would have already been allocated to that facility instead. This is an interesting and powerful property which is also given and proved in Mladenovic *et al.* (2003).

We take this observation into account to define our neighbourhood for attracting facilities. This is achieved by exploring those regions defined by RC_{jl} as the regions where a facility could be located; $j = 1,..., p$ and $l \in CP_j$.

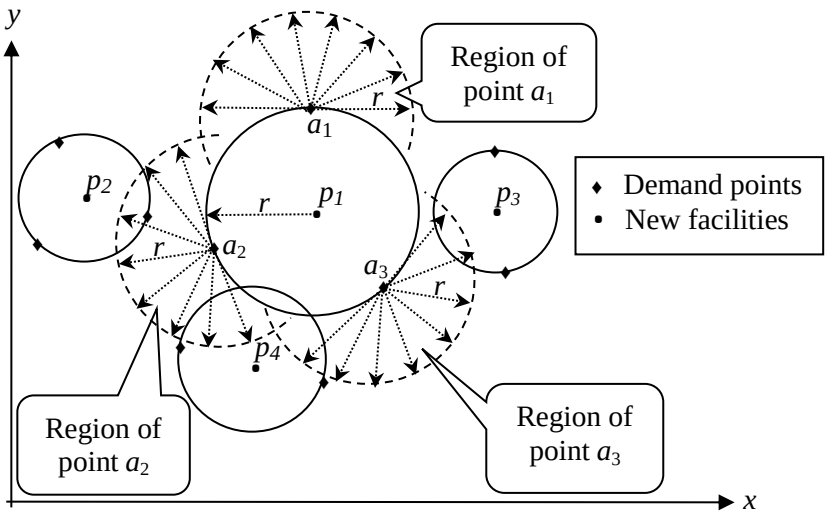

Figure 4: An example of 3 regions that do not contain any facility for a circle defined by 3 critical points a_1, a_2 and a_3

4.2 Facility Removal

In the k^{th} neighbourhood, instead of removing k facilities randomly from X, we remove these facilities from $CC_{k'}$ where k' is the level at that iteration, $k' = 1,..., p$, see Figure 5 for an illustration. The way k' is updated is defined next.

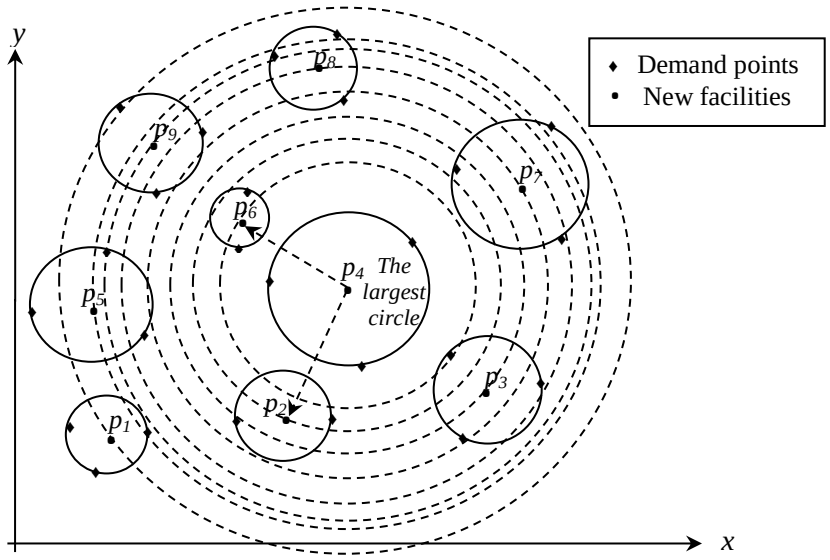

Figure 5: An example of the levels of covering circles that are dynamically increasing from the source region

The new neighbourhood

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

$$FN_k(X) = X \setminus \bigcup_{r=1}^{k} X_r \vee \bigcup_{r=1}^{k} X_r^{'}; k = 1,..., K_{\max} \quad (4)$$

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

The updating of $CC_{k'}$

As the removal process of the k facilities and their insertion is 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 which will also be given in the algorithm that follows in Figure 6. We first remove a facility from CC_1 namely the facility encompassed by the largest circle, this facility is then located randomly in UCR_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 UCR_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 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 $k \ge k'$, we reset $k = k'$ and $Flag = 1$. As we start with CC_1 we initialise $Flag$ to 1. Based on the neighbourhood structure described earlier and the way $CC_{k'}$ is updated, the new VNS(FN) algorithm is summarised in Figure 6.

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

$FN_k; k = 1, ..., K_{\max}$ using (4)

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

$k = k' = 1$ and $Flag = 1$

Step 2: While $k < K_{\max}$ do

- *Step* 2a: *(i)* If $k > k'$ set $k' = k$ and Flag=+1
  (ii) Generate $X' \in FN_k(X)$ using (4) ***"Shaking Part"***
- *Step* 2b: apply the Elzinga-Hearn algorithm or the enhanced versions for the affected clusters ***"Continuous locations"***
- *Step* 2c: Apply a local search to obtain X'' starting from X' ***"Local Search Part"***
- *Step* 2d: ***"Evaluation Part"***

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

  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: Record Time.

If $Time > CPU_{\max}$ we record the incumbent solution (X) and stop
Else set $k = 1$ and go to step 2.

Figure 6: The new VNS2(FN) algorithm

As an example in Figure 5, from CC_1 (the largest circle) we select its facility p_4 to locate randomly in UCR_1 (in one of the regions RC_{1l} with l being one of the critical points). 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_4 and p_6. These will be located randomly in the continuous space of UCR_2.

5 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.

5.1 Allocate a critical point of the largest circle to another facility

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 7.

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.

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 $j' = j$

elseif $R_j < R'$ 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 7: The allocation procedure (ALLOC)

Figure 8 (*a*) shows $C_{a_1}^{'}$ and $C_{a_1}^{''}$ based on the critical point a_1, initially served from facility p_1.

There are three facilities p_2, p_3 and p_4 in the region of V_{a_1}.

Allocating a_1 to one of these three facilities can improve the solution as long as the radius of the destination cluster is less than $R_{\max}$. Figure 8 (*b*) shows the case where the critical point a_1 is allocated to facility p_2 yielding a new radius $R_{\max}^{'} < R_{\max}$.

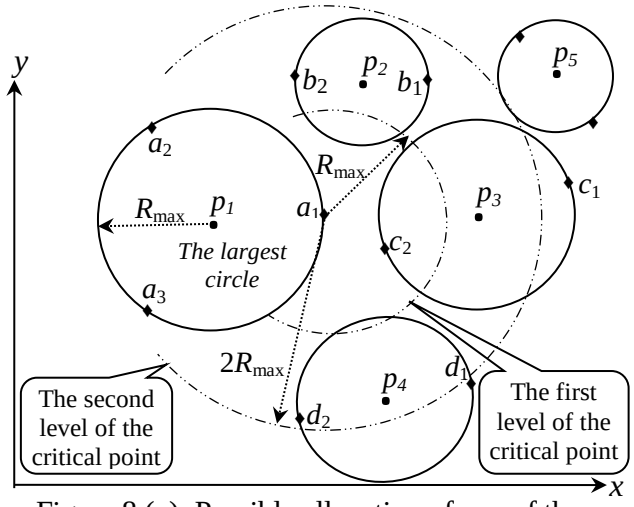

Figure 8 (*a*): Possible allocation of one of the critical points a_1 of the largest circle

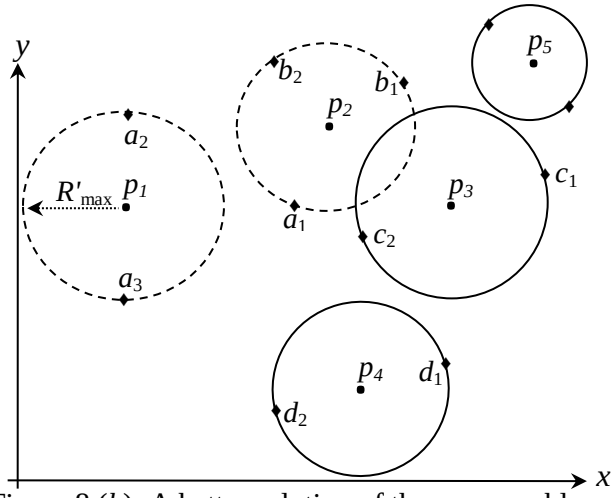

Figure 8 (*b*): A better solution of the same problem by allocating a critical point of the largest circle

To illustrate the impact of this reallocation, computational results of the Multi-Start algorithm using 1000 runs with and without this scheme are given in Table 3. The existing data set with known optimal solutions ($n = 439$ TSP-Lib) with $p = 10$ to 100 is used here. 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.

Table 3: Effect of the reallocation (based on 1000 runs of Multi-Start)

n= 439 TSP-Lib	Multi-Start		Multi-Start + Reallocation			
			Objective function & CPU		Improvement Deviation (%)	
p	Z	CPU Time	Z	CPU Time	Z	CPU Time
10	1803.120	55.264	1753.080	55.280	2.775	0.029
20	1140.290	77.219	1125.280	77.260	1.316	0.053
30	975.000	91.129	975.000	91.129	0	0
40	822.344	123.082	760.345	123.124	7.539	0.034
50	739.193	133.444	698.771	133.546	5.468	0.076
60	635.044	146.088	570.088	146.267	10.23	0.123
70	570.088	160.304	570.088	160.398	0.000	0.059
80	570.088	168.239	542.707	168.304	4.803	0.039
90	570.088	175.736	570.088	175.737	0	0.001
100	503.271	196.791	437.679	196.989	13.03	0.101
Average					**4.516**	**0.051**

5.2 Removal of the non promising facilities

The idea is to identify those facilities that serve the critical fixed points only and to allocate them to other facilities which will lead to such facilities having no customers and hence a reduction in the number of facilities. These saved facilities could then be located in the continuous space encompassed by the larger circles.

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

Let q be the number of facilities saved. These q facilities are then located one at time near the critical points of the largest circle based on the reallocation scheme described in Figure 8. The main steps of this removal procedure are summarised in Figure 9.

For instance, Figure 10 (*a*) shows a feasible solution of a 5-centre problem. Here the critical points of the circle centered 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 presented in Figure 10 (*b*), where the new $R_1 = R_{\max} = R'_{\max}$. The facility initially located at p_3 can now be relocated in the largest circle centered at p_1 leading to having two facilities, each with a radius $\leq R_{\max}$.

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 9: The removal procedure of the non-promising circles

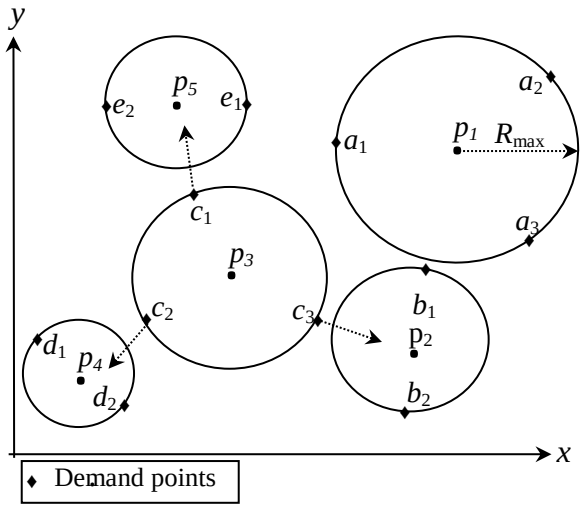

Figure 10 (*a*): A feasible solution of a 5-centre location problem (*removal of the facility at location* p_3)

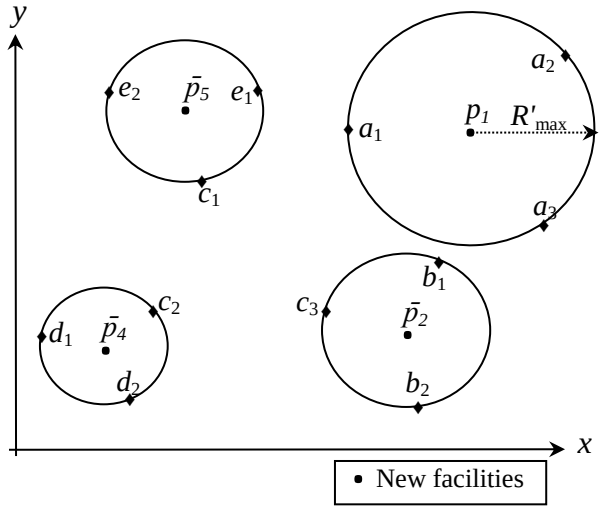

Figure 10 (*b*): The same objective function value but for its corresponding *4*-centre location problem (*Step* 2 of Figure 8)

Table 4 shows the computational results of this enhancement, when it is applied on the solutions of the multi-start algorithm with 1000 runs using the existing TSP data (n=439 TSP-Lib).

Table 4: Results of the Multi-Start for 1000 iterations with and without the removal-based enhancement

n= 439 TSP-Lib	Multi-Start		Multi-Start + Facility Removal procedure				# Saved Facilities
			Objective function & CPU		Deviation (%)		
p	Z	CPU Time	Z	CPU Time	Z	CPU Time	
10	1753.080	48.086	1753.080	48.417	0	0.688	0
20	1226.020	71.793	1226.020	71.840	0	0.065	0
30	975.000	92.004	975.000	92.029	0	0.027	0
40	975.000	107.657	975.000	109.056	0	1.299	0
50	834.742	141.196	822.344	141.417	-1.485	0.157	1
60	655.386	167.267	631.495	167.538	-3.645	0.162	1
70	580.005	175.504	503.271	176.015	-13.230	0.291	3
80	570.088	178.622	459.619	179.251	-19.380	0.352	5
90	570.088	190.107	459.959	190.944	-19.320	0.440	6
100	503.271	192.665	332.838	194.025	-33.870	0.706	7
Average					**-9.092**	0.419	**2.3**

5.3 The VNS(FN) algorithm

Our VNS(FN) algorithm is the VNS2(FN) algorithm described in Figure 6 with the local search in step 2c incorporating the ALLOC procedure of Figure 7 and the removal mechanism given in Figure 9.

6 A VNS with Memory

In the traditional VNS-based implementations the search is memoryless. In this section we intend to incorporate learning within the search so to identify any useful values of the parameters that are worth controlling in VNS.

6.1 The Algorithm

The learning consists of two stages. In the first stage, we record some information about the the progress of the VNS. This is performed during a certain time period defined as β% of the maximum CPU. The information that we are interested in includes the use of the k^{th} neighbourhood, the level of the coverage k' and the value of $K_{\max}$. The second phase uses the information obtained to guide the search in subsequent iterations of the VNS. A skeleton of the VNS with memory is given in Figure 11.

In *Step* 2 we can gather the information by recording the score of each neighbourhood either as a fraction of its use, or just the number of success. A density function say

$$\Pr ob(k) = \frac{Score(k)}{\sum_{k=1}^{K'_{\max}} Score(k)} \quad (5)$$

can then be computed with $K'_{\max}$ defining the initial value of $K_{\max}$.

The new value of $K_{\max}$ could be set as

$$K_{\max} = |\{k \in \{1,...,K'_{\max}\} : \Pr ob(k) > 0\}|. \quad (6)$$

Step 1: Set $CPU_{\max}, \beta, K'_{\max}$, $FN_k; k = 1,...,K'_{\max}$ and generate an initial solution

Step2 (Learning phase):

(i) Apply any variant of VNS during $\beta CPU_{\max}$ and record $Score(k)$ for each neighbourhood structure $k; k = 1,...,K'_{\max}$, and any other useful information.

(ii) Compute $\Pr ob(k); k = 1,...K'_{\max}$ using (5) and calculate $K_{\max}$ using (6).

Step 3 *(Adaptive VNS)*:

Use the information gathered in step 2 (ii) for the remaining iterations of VNS.

Figure 11: The VNS(FN) with Memory (VNS-M)

6.2 Application of VNS-M to the planar p-center problem

The facility-based neighbourhood can incorporate the process of learning by identifying the number of the preselected facility candidates (*k*) and the levels of the covering circles. Note that the customer-based neighbourhood method does not have such a flexibility as the value of k is fixed to 1, 2 or 3, representing the number of critical points and also the source region is fixed being defined by the largest circle. Since VNS(FN) is found to be the best performer, the learning process is carried out using this variant only.

Phase I: Learning process (Steps 1 & 2 of VNS-M)

Here, we use β = 0.25 for simplicity. We observe VNS(FN) behaviour by recording the information mentioned above.

The levels of the covering circle (k')

As the chosen facility is found by dynamically changing the radius of the covering circle $CC_{k'}; k' = 1,..., p$, the level k' is identified whenever a better solution is found. In other words, if there is an improvement at a given $CC_{k'}$, the frequency of using such a level will be increased by one.

The neighbourhood structure (k)

We record the number of times the solution is improved using the k^{th} neighbourhood structure; $k = 1,..., K_{\max}$ (k facilities are removed and inserted somewhere else according to our previous strategies). Furthermore, as part of the process we also derive $K_{\max}$ accordingly.

Phase II: Integrating the information within the search (*Step* 3 of VNS-M)

The information that is recorded in the first phase (the value of $k'; k' = 1,..., p$ and the value of $k; k = 1,..., K_{\max}$) is then used to guide the search in VNS(FN). Two schemes are explored:

The range (min, max)

As the size of the covering circle is dynamic, we would like to determine the maximum level that has achieved improvement. The same idea is also applied to fix the range for the value of k, i.e. $[a, b]$. Note that in the classical VNS, a = 1 and b=$K_{\max}$ whereas here though $a = 1$, b is not necessarily $K_{\max}$. However, in some cases, it was observed that the values of k' and k can be further away from their respective means than what is deemed reasonable (outliers), those that lie beyond the mean + 2standard deviations. Therefore, such outliers are excluded from our analysis.

A preliminary study shows 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 have the same level of importance. It was however observed that some levels improve 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.

The frequency of occurrence

The idea is to choose $\alpha \in (0,1)$ uniformly and compute $\hat{L} = F^{-1}(\alpha)$ with $F(L) = \sum_{t=1}^{L} \Pr ob(t)$

where $\Pr ob(t)$ refers to the probability of choosing the t^{th} level $(t = 1,..., p)$ or the t^{th} neighbourhood $(t = 1,..., K_{\max})$ and is computed using (5). 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 12 illustrates how such a scheme can be used.

This technique is also referred to, in the literature, as the inverse method. This method is more adaptive as both the values of k and k' are pseudo-randomly selected.

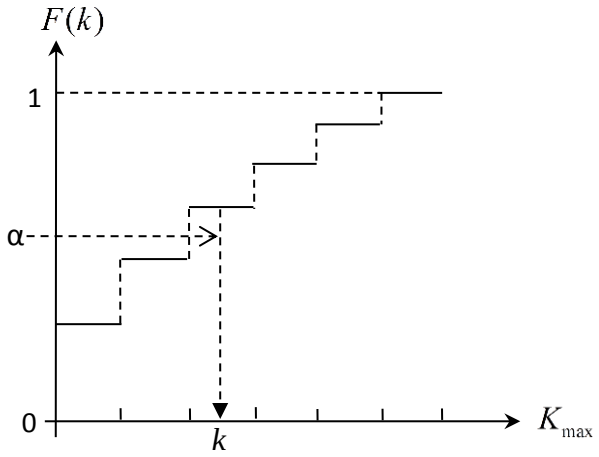

Figure 12: Selection of k using the frequency of occurrence

Some Comparative results

A preliminary experiment using both schemes on a TSP data set with n = 439 and p varying from 10 to 100 in steps of 10 is given in Table 5. The results based on 10 runs show that applying this scheme is more efficient than the range-based. For instance, the overall average deviations for the best results are 0.80% and 1.15%, with the average results being 1.96% and 2.65%. The ST Deviation values of 6.46 and 3.17 of schemes 1 and 2 respectively, also confirm that the frequency-based scheme is more reliable especially for large values of $p\,(eg;\ p \geq 30)$.

7 Computational Experiments

The proposed heuristics are 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 case, an integrated C++ code, with CPLEX incorporated within it, is used and run in the same computer. Our enhancements are 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 compare the computational results of our VNS based approaches to the optimal solutions provided by Chen and Chen (2009). For the other larger data sets no optimal solutions are available. To assess the performance of our approach, we used an efficient implementation of the set covering-based approach that optimally solves the vertex p-centre problem, as will be explained later. The optimal discrete solutions are then refined in the continuous space by applying the same local search as described in this paper. We run the multi-start procedure 10000 iterations and select the best solution. For consistency, we use the corresponding CPU time for the multi start as a stopping criterion in our VNS methods.

Table 5: Deviation (%) average and best results for the VNS-M using the range and the frequency-based technique using 10 random runs

n = 439	The optimal solutions (Z)	Using the range			Using the frequency of occurrence		
p		Deviation % Average Results	Deviation % Best Results	ST DEV	Deviation % Average Results	Deviation % Best Results	ST DEV
10	1716.510	0	0	0	0	0	0
20	1029.710	0	0	0	0	0	0
30	739.193	0.29	0	6.69	0	0	0
40	580.005	1.42	0	12.55	0.45	0	8.33
50	468.542	3.09	0	12.38	2.69	0.94	4.04
60	400.195	4.20	1.98	7.37	2.92	0.85	6.17
70	357.946	1.58	1.23	3.28	1.42	1.27	1.62
80	312.500	6.73	2.45	11.74	5.38	1.98	4.27
90	280.903	3.28	2.35	4.65	2.83	1.69	2.25
100	256.680	5.95	3.54	5.95	3.90	1.30	4.00
Average		2.65	1.15	6.46	**1.96**	**0.80**	**3.17**

7.1 Comparisons against Existing results (small data set)

For simplicity and ease of repeatability, the initial solution in our VNS-based heuristics is taken as the solution of the multi-start algorithm with 100 runs. In Table 6, the results for VNS(CN) and VNS(FN) with and without memory are reported. Our experiments show that both VNS heuristics (CN and FN) produce better results than the multi-start heuristic as well as the optimal solution based on the discrete case. In brief, the performance of VNS(CN) was slightly inferior to the VNS(FN) memoryless as the overall average deviation values from the optimal solutions are 0.429% and 0.362% respectively. It can be seen that VNS(FN) with memory is more effective, as the overall deviation has been reduced to 0.233%.

The optimal solutions are found by Chen and Chen (2009) who used an interesting relaxation method based on solving a succession of small sub-problems. The authors add a number of demand points each time the obtained optimal solution of the sub problem happens to be not feasible for the entire problem. For simplicity, in our subsequent tables, we refer to these values by k as reported in their paper.

Table 6: Deviation (%) of VNS(FN) (with and without memory) and CN from the optimal solution

n	*p*	The optimal solutions	Multi-Start' for 10 000 iterations	Neighbourhood Customer-based VNS(CN)	Neighbourhood Facility-based VNS (FN)	
					Memoryless	With Memory (VNS-M)
		Z	Deviation %	Deviation %	Deviation %	Deviation %
439	10	1716.5099	2.016	**0**	**0**	**0**
	20	1029.7148	11.417	**0**	**0**	**0**
	30	739.19297	31.901	**0**	**0**	**0**
	40	580.00539	18.062	**0**	**0**	**0**
	50	468.54162	29.412	0.674	**0.284**	0.674
	60	400.19527	42.452	**0.349**	0.845	**0.349**
	70	357.94553	59.267	1.272	**0**	**0**
	80	312.5000	30.703	1.203	1.203	**0.017**
	90	280.90256	25.928	**0.395**	**0.395**	**0.395**
	100	256.68019	27.457	**0.395**	0.896	0.896
	Average		27.862	0.429 (4)	0.362 **(5)**	**0.233 (5)**

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

7.2 Results on larger data set

Four larger datasets (*n*= 575, 783, 1002 and 1323 TSP-Lib) are used to assess the performance of our enhancements, see Table 7. As no optimal solution is available for these cases, we compute the deviation from the best solution as Deviation (%) = $100\frac{(Z_H - Z_{best})}{Z_{best}}$ 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. To provide additional comparisons, two strategies for generating the initial solution are proposed:

a) The solution of the multi-start procedure with 100 runs – Here, we use the solution of the multi-start algorithm with 100 runs, as previously shown in Table 6.

b) The optimal solution of the vertex-centre problem – The idea here is to determine the optimal solution of the vertex *p*-centre problem as a starting point using the set covering-based approach mentioned earlier.

In general, Table 7 shows that the performance of VNS(FN) with memory namely VNS-M outperforms all the others, yielding 34 best solutions in total, (i.e., 20 obtained using the solution of the multi-start and 14 with the optimal solution of the discrete problem). The CN-based approach achieved the best solution 13 times using the multi-start and 4 times with the discrete case (17 in total).

Table 7: Deviation (%) of VNS(FN) (with and without memory), VNS(CN), Multi-Start and the optimal solution based on the discrete case

n	*p*	Overall best solutions *Z*	Multi Start (10 000 Iterations)	VNS(CN)	VNS(FN)		Optimal Discrete solutions	Discrete + Continuous	Using Discrete –based initial solutions		
					Memory less	With Memory (VNS-M)			Discrete + Continuous + VNS(CN)	Discrete + Continuous + VNS(FN) Memory less	Discrete + Continuous + VNS(FN) With Memory (VNS-M)
575	10	67.9258	1.910	0.998	0.998	**0**	6.984	2.484	0.998	0.998	**0**
	20	45.6219	3.096	**0**	0.745	**0**	7.939	3.217	1.025	0.596	**0**
	30	35.5563	9.050	**0**	**0**	0.156	10.833	5.815	3.862	0.503	2.823
	40	30.2648	14.507	1.646	1.264	**0**	10.032	6.854	1.801	1.908	2.056
	50	26.1731	17.837	0.365	2.396	1.131	12.432	6.175	2.396	2.805	**0**
	60	23.6215	18.706	2.518	**0**	0.293	14.303	10.395	2.719	3.243	0.844
	70	21.0586	14.756	2.124	1.773	**0**	17.567	11.214	2.716	2.908	3.137
	80	19.5576	24.958	0.167	1.878	1.492	19.365	6.719	0.036	2.899	**0**
	90	17.9234	23.566	0.814	2.374	**0**	22.360	14.612	0.814	2.308	3.486
	100	16.6208	28.514	0.541	0.46	0.467	24.031	14.473	2.458	2.334	**0**
	Average		15.690	0.917	1.189	**0.3539**	14.585	8.196	1.883	2.050	1.235
783	10	79.3127	**0**	**0**	**0**	**0**	5.262	4.558	**0**	**0**	**0**
	20	53.4605	2.713	0.429	0.685	**0**	6.340	6.118	0.919	1.013	0.429
	30	42.3949	11.837	**0**	2.063	0.494	8.657	2.986	0.949	1.525	1.871
	40	35.9619	10.751	1.591	**0**	0.411	10.005	6.157	14.74*	14.74*	14.74*
	50	31.4086	15.17	0.19	0.87	**0**	10.750	5.404	31.38*	31.38*	31.38*
	60	28.0533	18.185	**0**	0.036	1.098	11.930	7.892	23.55*	23.55*	23.55*
	70	25.4456	20.885	**0**	1.568	0.694	13.356	9.633	6.977	8.519	4.643
	80	23.5601	22.668	0.845	0.057	**0**	14.282	10.723	9.557	2.418	0.697
	90	21.7099	24.708	1.572	3.672	**0**	17.435	12.946	5.208	3.373	1.806
	100	20.334	26.014	1.086	2.026	**0**	18.231	12.252	6.759	3.672	1.086
	Average		15.293	0.571	1.098	**0.2697**	11.625	7.8669	10.004	9.019	8.020
1002	10	2389.36	0.889	**0**	**0**	**0**	6.312	4.031	**0**	**0**	**0**
	20	1609.54	4.662	**0**	1.292	1.29	7.252	4.194	1.29	1.29	0.122
	30	1231.36	8.418	0.108	2.02	**0**	9.334	3.131	0.108	0.801	1.18
	40	1030.4	17.064	1.299	1.352	**0**	13.698	5.702	2.994	1.352	2.047
	50	906.228	16.389	**0**	1.141	0.193	13.609	10.949	**0**	0.962	1.217
	60	801.474	21.130	0.153	2.196	**0**	13.842	8.099	2.216	1.027	1.324
	70	727.154	17.700	0.976	1.291	**0**	16.894	11.407	1.619	2.047	**0**
	80	664.798	22.029	1.722	2.298	1.046	14.558	5.563	1.046	1.046	**0**
	90	604.494	28.273	**0**	1.725	0.745	18.428	8.420	0.965	2.728	4.134
	100	559.017	29.425	2.078	3.73	2.078	20	8.074	3.73	3.630	**0**
	Average		16.598	0.634	1.705	**0.5352**	13.393	6.957	1.397	1.488	1.002
1323	10	2897.49	0.327	0.237	0.067	0.067	6.206	1.735	0.067	0.067	**0**
	20	1886.82	4.414	**0**	**0**	**0**	6.868	4.602	**0**	**0**	**0**
	30	1466.97	8.293	1.622	2.673	0.984	11.216	5.967	0.681	1.927	**0**
	40	1236.38	12.405	**0**	0.343	1.210	9.381	4.986	0.343	0.631	0.343
	50	1060.82	15.989	**0**	1.458	0.420	11.920	8.016	1.229	0.841	1.458
	60	941.87	12.663	1.227	2.194	**0**	12.862	6.481	0.738	0.464	1.925
	70	844.967	19.382	0.934	1.615	**0**	15.025	11.329	2.119	1.016	0.491
	80	774.764	15.335	1.092	2.646	**0**	15.526	11.745	2.183	2.614	1.644
	90	720.625	24.000	0.661	**0**	2.117	15.455	10.936	4.787	0.34	0.204
	100	662.936	28.633	2.237	1.662	5.129	18.729	10.774	2.494	1.649	**0**
	Average		14.144	0.801	1.266	0.993	12.319	7.657	1.464	0.955	**0.607**
Overall Average			15.431 (1)	0.731 (13)	1.314 (7)	**0.538 (20)**	12.965 (0)	10.102 (0)	7.657 (4)	3.378 (3)	2.716 (14)

( ): *The number of cases when the best solution is found.* Bold: *The best solutions found.* *: *No VNS due to CPU time.*

The memoryless VNS(FN) obtained 7 and 3 times the best out of 40 for strategies (*a*) and (*b*) respectively. It can also be observed that the optimal solution based on the discrete case fails to find even one best solution, while the multi-start algorithm (10000 runs)

achieved the best solution only once. In addition, the average deviation values also confirm that the performance of VNS(FN) with memory always yield relatively better results than those of the other enhancements, with an overall average deviation of 0.538 % and 2.716% when using the solution of the multi-start and the optimal discrete solution respectively. These compare favourably with (0.731 %, 7.657%) and (1.314 %, 3.378%)) for VNS(CN) and VNS(FN) without memory respectively. Note that when p = 40, 50 and 60 with n = 783 there was no remaining time to run the VNS when the optimal discrete solutions was used as the initial solutions on its own consumed more time than required by the multi start.

In brief, we can confirm that the performance of the VNS(CN) is better than the VNS(FN) without memory, but the incorporation of learning into the search has made VNS(FN) with memory to be the best performer.

7.3 Time performance

A comparison between the average total CPU time of the Multi-Start algorithm (10000 iterations) and the average CPU time when the best continuous solution is found for both cases (using the solution of the multi-start procedure with 100 runs and the optimal discrete solutions as the initial solutions) as well as Chen and Chen's results (when it is available) is presented in Table 8. It is worth noting that the recording of when the best solution is obtained could be useful in designing a more advanced stopping rule. To achieve this, we record the CPU time when the best solution is found by a given heuristic as T_H and compute the deviation from the CPU time required for 10000 iterations of the multi-start algorithm which we refer to as T_{MS} . Deviation is computed as follows:

$$Deviation(\%) = 100\frac{(T_H - T_{MS})}{T_{MS}}$$

To provide a fair comparison in terms of CPU, we use 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 by Chen and Chen (2009) cannot be easily identified for the number of Mflops, we provide an approximate time using a slightly slower but similar computer namely a PC Intel Pentium 4 (3.06 GHz), 2 GB of main memory.

Table 8 shows that the overall deviations of CPU time when the best solution is found to increase with *n* for all the algorithms. For instance, in VNS (CN), when using the solution of the multi-start (100 runs) as the initial solution, the overall deviations vary from nearly -82% for *n*=439 to nearly -40% for *n*=783.

In general, it can be seen that applying VNS(FN) and VNS(CN) require around 50% of the time required by the multi start algorithm.

Table 8: Average CPU time of the Multi-Start algorithm (for *p*=10 to 100 in increment of 10), Deviation (%) of CPU time for VNS(FN) (with and without memory) and VNS(CN)

n	Deviation (%)								
	Average total CPU time (10000 iterations) (secs)	Initial solution based on 100 restarts			Initial solution based on Discrete-based			Chen and Chen's results (Continuous Solutions)	
		VNS(CN) (Best CPU Time)*	VNS(FN) (Best CPU Time)*		VNS(CN) (Best CPU Time)*	VNS(FN) (Best CPU Time)*		Improved relaxation (*k*=7)	Binary relaxation (*k*=6)
			Memory less	With Memory (VNS-M)		Memory less	With Memory (VNS-M)		
439	1497.56	-81.73	-73.21	-74.64	-77.38	-65.11	-74.39	-88.37	-98.72
575	1681.81	-55.90	-47.52	-36.91	-31.59	-32.58	-17.22	N/A	N/A
783	2762.45	-39.85	-39.65	-48.84	-28.70	-17.55	-15.00	N/A	N/A
1002	4398.09	-45.43	-59.28	-57.98	-9.95	-44.12	-44.13	N/A	N/A
1323	5662.98	-50.79	-33.67	-48.44	-47.06	-41.83	-37.05	N/A	N/A
Average		-54.74	-50.67	-53.36	-38.94	-40.24	-37.56	N/A	N/A

*:*Time when the best solution is found*
k: the number of added demand points at each stage in the Chen and Chen's paper

7.4 An intensification of the local search in VNS (GVNS)

In this section we intensify the local search within VNS which is based on the allocation procedure 'ALLOC' of Figure 7. We achieve this by exploring not only one critical point of the largest circle at a time but also all the 3 pairs of critical points as well as all the three critical points simultaneously.

Adaptation of ALLOC

ALLOC is based on the following neighbourhood structure $N_1(X)$ which is the removal of one critical point $l \in CP_1$ and inserting it in the best region in V_l.

Here we extend this to cater for $N_2(X)$ and $N_3(X)$ to define the neighbourhoods that simultaneously remove all the possible 2 critical points (3 pairs) and the full triplet (3 critical points) from the largest circle. This is performed by solving the corresponding 1-center problem (*Step* 2 of ALLOC) on the reduced largest circle and allocating these critical points in their respective $V_l, l \in CP_1$ (*Step* 3 where the construction of V_l is carried out and *Step* 4 where the

choice is made). *Steps* 1 and 5 of ALLOC remain unchanged. In other words instead to choose the best move from at most 3 cases we now intensify the search by evaluating at most 7 cases.

We incorporate the above VND with the three neighbourhood structures N_1, N_2 and N_3 as our local search in *Step* 2c in both VNS(CN) and in our best variant of VNS(FN) namely VNS-M.

Table 9: Deviation (%) of VNS(FN) and VNS (CN) with and without VND

n	*p*	Overall Best solutions (*Z*)	Without intensified local search (VND)				With intensified local search (VND)	
			Initial solution based on 100 restarts			Initial solution based on Discrete case	Initial solution based on 100 restarts	
			VNS (CN)	VNS(FN) Memory less	VNS-M	VNS-M	VNS (CN)	VNS-M
575	10	67.9258	0.998	0.998	**0**	**0**	**0**	**0**
	20	45.6219	**0**	0.745	**0**	**0**	**0**	**0**
	30	35.5563	**0**	**0**	0.156	2.823	**0**	0.503
	40	30.2648	1.646	1.264	**0**	2.056	0.698	0.322
	50	26.1731	0.365	2.396	1.131	**0**	0.463	0.820
	60	23.6215	2.518	**0**	0.293	0.844	1.624	0.687
	70	21.0586	2.124	1.773	**0**	3.137	1.134	0.917
	80	19.5086	0.418	2.134	1.747	0.251	**0**	1.002
	90	17.9234	0.814	2.374	**0**	3.486	0.582	0.621
	100	16.5511	0.965	0.883	0.89	0.421	**0**	0.329
	Average		0.985	1.257	**0.422**	1.302	0.450	0.520
783	10	79.3127	**0**	**0**	**0**	**0**	**0**	**0**
	20	53.4405	0.466	0.723	0.037	0.466	1.051	**0**
	30	42.3949	**0**	2.063	0.494	1.871	0.456	0.213
	40	35.9619	1.591	**0**	0.411	14.74	1.227	0.274
	50	31.184	0.911	1.597	0.720	32.32	**0**	1.078
	60	28.0533	**0**	0.036	1.098	23.55	1.098	1.098
	70	25.4456	**0**	1.568	0.694	4.643	1.772	0.387
	80	23.5601	0.845	0.057	**0**	0.697	0.048	0.14
	90	21.7099	1.572	3.672	**0**	1.806	2.375	1.022
	100	20.334	1.086	2.026	**0**	1.086	0.813	0.689
	Average		0.647	1.174	**0.345**	8.119	0.884	0.490
1002	10	2389.36	**0**	**0**	**0**	**0**	**0**	**0**
	20	1609.54	**0**	1.292	1.29	0.122	**0**	0.567
	30	1231.36	0.108	2.02	**0**	1.18	1.180	0.210
	40	1030.4	1.299	1.352	**0**	2.047	1.154	1.276
	50	901.455	0.529	1.677	0.724	1.753	**0**	0.561
	60	801.474	0.153	2.196	**0**	1.324	1.603	0.745
	70	727.154	0.976	1.291	**0**	**0**	0.976	0.236
	80	664.798	1.722	2.298	1.046	**0**	1.256	1.256
	90	604.152	0.057	1.782	0.802	4.193	**0**	1.022
	100	559.017	2.078	3.730	2.078	**0**	2.078	2.078
	Average		0.692	1.764	**0.594**	1.062	0.825	0.795
1323	10	2897.49	0.237	0.067	0.067	**0**	0.237	0.067
	20	1868.92	0.958	0.958	0.958	0.958	0.88	**0**
	30	1466.97	1.622	2.673	0.984	**0**	2.107	1.408
	40	1236.38	**0**	0.343	1.21	0.343	**0**	1.009
	50	1060.82	**0**	1.458	0.42	1.458	0.700	0.841
	60	941.672	1.249	2.216	0.021	1.946	**0**	0.759
	70	844.967	0.934	1.615	**0**	0.491	1.016	1.306
	80	774.764	1.092	2.646	**0**	1.644	1.82	**0**
	90	720.625	0.661	**0**	2.117	0.204	0.227	1.004
	100	662.936	2.237	1.662	5.129	**0**	1.403	0.573
	Average		0.899	1.364	1.091	0.705	0.839	**0.697**
Overall Average # best			0.806 (10)	1.390 (6)	**0.613** **(16)**	2.797 (11)	0.749 (13)	0.626 (7)

Computational results

The same data sets and the same stopping criterion that were used in the previous computational results are also applied here. The overall results of the best variants investigated in this study are given in Table 9. Note that the results obtained by those methods that are found to be inferior and dominated by at least another method, such as the multi-start for instance, are not reported in this summary table. In particular, the results obtained by VND that starts from the optimal solution of the discrete problem are not reported as the results are almost always inferior. The new results are very competitive as 8 new best results were discovered (6 based on VNS(CN) and 2 with VNS-M) excluding 9 other already found best solutions. Also, the new variants are found to be better suited for the very large instance where an average of slightly less than 0.7% is achieved. In addition, very encouraging overall average deviations of 0.629% and 0.839% are recorded by VNS(CN) and VNS-M respectively which compare favourably against the best results found by VNS-M (i.e., 0.613%). This slight deterioration can be due to the time spent in VND which obviously slightly limit the exploration of VNS given that the same computational time of 10,000 multi-starts is used.

7.5 Brief results on the Discrete case

For completeness we also provide the optimal solutions for the vertex *p*-centre problem for these large instances using the set covering-based approach, see Table 10. This is based on Daskin (1995) algorithm but incorporates an efficient data structure for sorting the useful elements of the distance matrix that are used during the search (see Al-Khedhairi and Salhi (2005)) besides starting with tighter initial upper and lower bounds as suggested by Salhi and Al-Khedhairi (2010). This basic enhancement speeds up the convergence considerably as it considers the existing distance elements of the distance matrix only as well as it identifies empty gaps between successive distances including the final empty gap between the last upper and lower bounds.

This is also relatively more efficient than the Chen-Chen algorithm which is sensitive to the number of demand points added (the *k* value in their paper). For instance, when $n = 1323$ the average CPU time are 227.27 and 1188.83 seconds respectively. The current approach also reduces the number of Cplex calls by almost half, which is significant when compared to the original implementation of Daskin (1995).

Table 10: The optimal solution, CPU time and number of calls to Cplex

TSP-Lib	P	*Z*	Discrete-based		Chen and Chen's results (Discrete Solutions)	
			CPU Time	# Cplex calls	Improved relaxation (*k*=13)	Binary relaxation (*k*=11)
439	10	1971.830	2.61	12	0.46	0.26
	20	1185.590	2.51	13	3.57	1.19
	30	883.529	2.75	11	N/A	N/A
	40	671.751	2.73	10	2.62	0.49
	50	564.025	3.22	10	N/A	N/A
	60	500.000	3.75	9	N/A	N/A
	70	474.341	3.78	9	N/A	N/A
	80	410.030	4.02	10	N/A	N/A
	90	395.284	3.02	7	N/A	N/A
	100	350.000	4.08	10	N/A	N/A
Average				10.1	N/A	N/A
575	10	72.670	61.16	11	N/A	N/A
	20	49.244	63.54	11	N/A	N/A
	30	39.408	1081.12	12	N/A	N/A
	40	33.301	363.24	12	N/A	N/A
	50	29.427	509.12	11	N/A	N/A
	60	27.000	269.88	10	N/A	N/A
	70	24.758	364.15	12	N/A	N/A
	80	23.345	267.35	12	N/A	N/A
	90	21.931	93.84	9	N/A	N/A
	100	20.615	32.68	10	N/A	N/A
Average				11	N/A	N/A
783	10	83.486	11.22	11	N/A	N/A
	20	56.850	429.46	11	N/A	N/A
	30	46.065	1941.59	10	N/A	N/A
	40	39.56	4164.56	9	N/A	N/A
	50	34.785	5567.42	8	N/A	N/A
	60	31.400	7610.13	10	N/A	N/A
	70	28.844	3029.35	10	N/A	N/A
	80	26.925	1412.01	11	N/A	N/A
	90	25.495	975.77	9	N/A	N/A
	100	24.041	304.06	13	N/A	N/A
Average				10.2	N/A	N/A

n TSP-Lib	p	*Z*	Discrete-based		Chen and Chen's results (Discrete Solutions)	
			CPU Time	# Cplex calls	Reverse relaxation (*k*=16)	Binary relaxation (*k*=16)
1002	10	2540.180	13.37	13	N/A	N/A
	20	1726.270	79.39	11	N/A	N/A
	30	1346.290	29.53	14	N/A	N/A
	40	1171.540	229.17	14	N/A	N/A
	50	1029.560	64.95	12	N/A	N/A
	60	912.414	10.12	11	N/A	N/A
	70	850.000	9.85	11	N/A	N/A
	80	761.577	6.55	11	N/A	N/A
	90	715.891	7.12	10	N/A	N/A
	100	670.820	7.20	12	N/A	N/A
Average				11.9	N/A	N/A
1323	10	3077.300	100.98	13	115.63	15.76
	20	2016.400	101.93	17	255.09	88.80
	30	1631.500	138.21	13	1028.47	341.53
	40	1352.360	324.86	14	988.99	965.10
	50	1187.270	400.96	14	3366.75	7168.02
	60	1063.010	829.02	13	5232.12	5104
	70	971.925	89.21	12	677.49	657.48
	80	895.055	120.51	15	118.08	588.73
	90	832.000	78.77	10	62.24	365.10
	100	787.095	88.23	12	43.46	334.71
Average				13.3	1188.83	1562.92
1817	10	457.905	906.10	14	65.64	N/A
	20	309.014	7120.13	17	5281.43	N/A
	30	240.987	4807.74	14	11398.47	N/A
	40	209.436	42066	10	N/A	N/A
	50	184.905	72811	13	N/A	N/A
	60	162.637	71249	14	N/A	N/A
	70 +	NF	NF	NF	N/A	N/A
Average *				15.0	5581.85	N/A

NF: *Not Found within 24 hours of CPU*

**: based on the first 3 instances (p=10, 20, 30)*

k: the number of added demand points in the Chen and Chen's paper

8 Conclusion and Suggestions

A VNS-based approach is designed to solve the *p*-centre problem on the plane which seems to have not attracted as many researchers as one may wish especially for larger instances. A local search which is similar to Cooper's algorithm is used. Enhancements on the well known Elzinga-Hearn algorithm for the *1*–centre problem are presented that produced nearly 60% reduction in CPU time. This is then embedded as part of the local search in VNS for solving the *p*-centre problem. Two modifications are proposed in our local search (allocation phase) as well as new neighbourhood structures designed for this particular location problem. The idea of incorporating learning within the search on the

best enhancement namely VNS(FN) proved to be very effective. This VNS with memory can be considered as a new adaptive VNS variant which can be easily implemented in many other combinatorial and global optimisation problems. To intensify the search within the VNS, a VND type mechanism is embedded into the local search which generated interesting new best results. The multi-start procedure is adopted for comparison purposes and its computing time used as a basis. The optimal solution of the vertex *p*-centre problem (the discrete case) is also found using an efficient implementation of the set covering based approach which is then refined for the continuous space by the same local search. Four TSP data sets with n = 439, 575, 783, 1002 and p varying from p =10 to 100 with a step of 10 are used as a platform to test our methodology. To our knowledge, this is the first time such larger instances were attempted. In summary, the VNS that uses facility-based and memory proved to be the best performer and the most robust when compared to the other methods.

For future research, it may be interesting to incorporate into our approach the optimal method given by Drezner (1984*a*) for the case of $p = 2$, whenever two clusters are considered worth solving optimally. In this study, we explored the removal of the non-promising facilities by identifying those facilities that serve the critical fixed points only, this can be relaxed to also consider those facilities that also have, in addition to the critical point, a small number of non-critical fixed points. Other related location problems with different objective function such as the Min Max Sum or other types of covering (maximum or partial covering on the plane) can also be investigated using our methodology. The methodology can also be extended to cater for area coverage which may or may not be convex as attempted by Wei *et al.* (2006). The use of memory within VNS, or in any other meta-heuristic that relies on certain parameters or on the sequence in which certain moves are implemented, could, in our view, be a challenging but a promising research avenue that is worthwhile exploring.

Acknowledgments- The authors would like to thank the anonymous referees for their interesting suggestions that improved the content as well as the presentation of the paper.

References

[1] Al-Khedhairi A and Salhi S. Enhancements to two exact algorithms for solving the vertex p-center problem, *Journal of Mathematical Modelling Algorithms*, 4: 129-147, 2005.

[2] Chen D and Chen R. New relaxation-based algorithms for the optimal solution of the continuous and discrete *p*-center problems. *Computers & Operations Research*, 36: 1646- 1655, 2009.

[3] Cooper L. Heuristic methods for location-allocation problem, SIAM Review 6, 37-53, 1964.

[4] Daskin MS. *Network and discrete locations: models, algorithms and applications*, John Wiley & Sons, NY, pp. 154-191, 1995.

[5] Dongarra JJ. Performance of various computers using standard linear Equation software. http://www.netlib.org/benchmark/performance.pdf (Accessed online 15 April 2013).

[6] Drezner Z. The planar two-center and two-median problems. *Transportation Science* 18: 351-361, 1984*a*.

[7] Drezner Z. The p-center problem- heuristic and optimal algorithms. *Journal of the Operational Research Society* 35: 741-748, 1984b.

[8] Eiselt HA and Charlesworth G. A note on *p*-center problems in the plane. *Transportation Science* 20: 130-133, 1986.

[9] Elzinga J and Hearn DW. Geometrical solutions for some minimax location problems, *Transportation Science* 6: 379-394, 1972.

[10] Hansen P, Mladenovic N, Brimberg J and Moreno-Perez JA. Variable neighbourhood search, in *Handbook of Metaheuristics*, (Gendreau M and Potvin J-Y , Eds), chapter 3, 61-86, 2010. (2nd edition), Springer, NY.

[11] Hearn, DW and Vijay J. Efficient algorithms for the (weighted) minimum circle problem. *Operations Research* 30: 777-795, 1982.

[12] Lu, C. Robust weight vertex p-centre model considering uncertain data: An application to emergency management. *European Journal of Operational Research* 230: 113-121, 2013.

[13] Megiddo N and Supowit KJ. On the complexity of some common geometric location problems, *SIAM Journal of Computing* 13: 182-196, 1984.

[14] Mladenović N and Hansen P. Variable neighbourhood search. *Computers & Operations Research* 24: 1097-1100, 1997.

[15] Mladenović N, Labbé M and Hansen P. Solving the p-center problem with tabu search and variable neighbourhood search. *Networks* 42: 48-64, 2003.

[16] Mladenović N, Todosijević R and Urosević D. An efficient General variable neighbourhood search for large TSP problems with time windows. *Yugoslav Journal of Operations Research* 23: 19-31, 2013.

[17] Murray AT and Wei R. A computational approach for eliminating error in the solution of the location set covering problem. *European Journal of Operational Research* 224: 52-64, 2013.

[18] Plastria F. Continuous covering location problem, *facility location: applications and theory* (Z. Drezner ed.), New York: Springer, 37–79, 2002.

[19] Salhi S and Al-Khedhairi. A integrating heuristic information into exact methods: The case of the vertex *p*-centre problem. *Journal of the Operational Research Society* 61: 1619-1631, 2010.

[20] Suzuki A and Okabe A. Using Voronoi diagrams, *Facility Location: A Survey of Applications and Methods*, (Z. Drezner ed.), New York: Springer, 103–118, 1995.

[21] Wei H, Murray AT and Xiao N. Solving the continuous space *p*-centre problem: planning application issues, *IMA Journal of Management Mathematics* 17: 413–425, 2006.

[22] Xu S, Freund R and Sun J. Solution methodologies for the smallest enclosing circle problem. *Computational Optimization and Applications.* 25: 283–292, 2003.