HEC MONTRÉAL

Affiliée à l'Université de Montréal

Variable Neighborhood Search Methods for the Dispersion Graph Problems, With Application to Franchise Location Problems

par

Behnaz Saboonchi

HEC Montréal, Méthodes Quantitatives de Gestion

Thèse présentée à la faculté des études supérieures et postdoctorales en vue de l'obtention du grade de Ph.D. en méthodes quantitatives

Mai 2013

HEC Montréal
Affiliée à l'Université de Montréal

Cette thèse intitulée :

Variable Neighborhood Search Methods for the Dispersion Graph Problems, With Application to Franchise Location Problems

présentée par
Behnaz Saboonchi

a été évaluée par un jury composé des personnes suivantes :

Professeur Patrick Soriano
Président-rapporteur

Professeur Pierre Hansen
Directeur de recherche

Professeur Sylvain Perron
Codirecteur de recherche

Professeur Gilles Caporossi
Membre du jury

Professeur H.A. Eiselt
Examinateur externe

Professeur James Loveland
Représentant du doyen de la FES

Résumé

Cette thèse porte sur le développement et l'analyse de diverses méthodes heuristiques basées sur la métaheuristique de recherche à voisinage variable (VNS) pour la résolution du problème de dispersion dans les graphes. La prise en compte de la dispersion dans les problèmes de localisation est sans conteste très pertinente pour modéliser les effets d'empiètement et de cannibalisation des revenus, deux préoccupations majeures des chaines de franchises. Les effets indésirables d'empiètement peuvent être pris en compte à travers les modèles de localisation en cherchant non seulement à maximiser la dispersion entre les nouvelles installations mais également la dispersion entre les nouvelles installations et les anciennes installations déjà présentes.

Le premier chapitre porte sur le problème MaxMinSum (p-dispersion-sum) qui consiste à choisir p installations parmi un ensemble d'installations possibles de manière à maximiser la plus petite somme de distances entre les installations choisies. Il s'agit de la première application d'une méthode heuristique pour cette variante du problème de dispersion. Des expériences numériques détaillées permettent de comparer les approches proposées pour prendre en compte l'intensification et la diversification des solutions explorées à l'intérieur de nos méthodes heuristiques inspirées de VNS.

Le deuxième chapitre est consacré au problème MaxSumSum (maximum diversity) dans lequel on choisit p installations de manière à maximiser la somme des distances entre toutes les paires d'installations choisies. Plusieurs méthodes heuristiques ont déjà été proposées pour ce problème et nous proposons une méthode de type VNS gloutonne. L'utilisation de nouvelles approches de recherche locale et de nouvelles fonctions de perturbation nous a permis d'améliorer plusieurs des meilleurs résultats connus.

Le troisième chapitre traite du problème MaxMinMin (p-dispersion) qui consiste à choisir p installations de manière à maximiser la plus petite distance entre toutes les paires d'installations choisies. La méthode proposée combine de nouvelles techniques de recherche et de perturbation pour contourner les effets de plateaux inhérents à ce type de problème. Il s'agit de la première application de VNS à ce problème.

Le quatrième chapitre propose un nouveau modèle multi-critères prenant en compte simultanément la dispersion entre les installations et la proximité des clients desservis, deux préoccupations majeures dans la localisation de franchises. Dans le modèle proposé, la dispersion entre les franchises est mesurée par le critère MaxMinMin alors que la proximité des clients est modélisée par un objectif de type gravité p-médiane où les clients ne sont pas nécessairement affectés à l'installation la plus proche mais plutôt affectés proportionnellement à une fonction d'utilité combinant l'effet attractif de l'installation et la distance.

Enfin, un survol des différentes méthodes est présenté dans la conclusion. Cette analyse confirme que le choix d'une méthode de perturbation astucieuse influence la qualité des solutions pour toutes les variantes du problème de dispersion. Par ailleurs, on montre que les méthodes exactes sont incapables de trouver des solutions à l'intérieur d'un niveau de tolérance acceptable sur l'optimalité dans un d'élai raisonnable, ce qui confirme la nécessité d'utiliser des méthodes heuristiques pour la résolution de problèmes de dispersion de taille raisonnable.

Mots clés : Optimisation combinatoire, Métaheuristiques, Recherche à voisinage variable, Localisation de franchises, Problèmes de dispersion, Modèles de gravité, Modèles multi-objectifs, Frontière de Pareto

Summary

This work explores various heuristics based on the Variable Neighborhood Search (VNS) metaheuristic framework for the dispersion graph problems. The first three chapters study the classical dispersion location models, whereas the fourth chapter focuses on a new multiobjective problem including a dispersion and a gravity p-median objective.

The idea to study the dispersion location problems was inspired from the encroachment and revenue cannibalization issues that are one of the main concerns of today's franchised chains. The undesirable effects of encroachment can also be captured from a different viewpoint in the sense that one can create multiple facility location models which are basically aimed at creating dispersed solutions by maximizing the dispersion among either the newly added units, or among the existing units and the new ones. To the best of our knowledge this work is the first application of various dispersion location problems within the franchise location context.

The first chapter addresses the MaxMinSum (p-dispersion-sum) problem that consists of the selection of p facilities among n candidate locations in a way that the smallest sum of the distances among the selected facilities is maximized. This is the first application of any heuristic to this variant of the dispersion problems and the extensive experiments present various intensification and diversification possibilities within the VNS framework. Comparisons with exact methods confirm the high quality of the proposed methods.

The second chapter studies the MaxSumSum (maximum diversity) problem that is relatively well explored by several heuristics in the literature. This problem maximizes the total sum of the distances between each pair of the located facilities. The proposed Greedy VNS heuristics including innovative local search and shake functions improve several results for the known benchmark test problems.

The third chapter addresses the MaxMinMin (p-dispersion) problem, which maximizes the smallest distance among all the selected facilities. The proposed elaborate `VNS` framework coupled with new plateau search and shake modules, is the first application of `VNS` to this problem. This method compares favorably with the existing state-of-the-art heuristics in terms of the quality and robustness of the obtained solutions, lower running times and computational complexity. Besides, several optimal solutions of the largest benchmark instances are found for the first time in the literature.

The fourth chapter presents an innovative bi-objective model which addresses the dispersion concept among the franchised units as well as the proximity to the customer zones. This is a new model that encompasses the most important factors in real life franchise location practices. The selected dispersion metric is the MaxMinMin criteria and the proximity to the clients is modeled by a gravity p-median objective.

Finally, an integrated overview of the proposed methods within the General Conclusion section confirms that the use of intelligent shake functions would significantly improve the quality of the solutions for all the dispersion problems. Besides, the exact methods are not capable of finding results within a reasonable percentage of the best known solutions that confirms the need for heuristic solution procedures to tackle challenging dispersion problem instances.

KeyWords : Combinatorial optimization, Metaheuristics, Variable Neighborhood Search, Franchise location, Dispersion problems, Gravity models, Multiobjective models, Pareto front

Table of Contents

List of Tables

List of Figures

Aknowledgements

This thesis would have been impossible without the guidance of my thesis supervisor Professor Pierre Hansen. I could not have asked for more inspirational, supportive and knowledgeable role model. You made this dream come true!

Professor Sylvain Perron, my dear co-supervisor! I made it to the end just in time, mostly because of your dedication and support. I have always admired your great personality, energy and work ethics and feel very privileged to have worked with you.

I would like to thank my colleagues and friends, specially Anthony Guillou who constantly helped me and shared his technical expertise during the five years we've been working together. I wish you the best of luck and am very thankful of all your help. I'd also like to thank the great people at GERAD and HEC Montréal specially Carole Dufour and Marie Perreault for having welcomed me and supported me when I first arrived in Montréal and was lost and could not speak French. I'm also very grateful of the support that Pierre Girard and François Guertin provided me during the tough moments of my thesis.

I would also like to thank my examiners, Professor Patrick Soriano, Professor Gilles Caporossi and Professor H.A. Eiselt for their constructive feedback and for turning my PhD defense into a memorable moment. Besides, I'm very thankful of Professor Gerard Cliquet and Professor Jean-Charles Chebat for helping me shape my thesis during its early stages and for their feedback.

Finally, I am dedicating this thesis to my lovely parents, my sister and husband for their unconditional love and support throughout my life!

General Introduction

Managing and locating retail and service networks have been widely addressed in different empirical and mathematical studies. The location decision process becomes more complex when it comes to franchise chains where the objective is not locating a single independent store, but a group of stores over time within a network. This location decision should be made without cannibalizing the existing same-brand units' revenues while maximizing the revenue and market share of the whole chain.

The undesirable effects of encroachment could be captured from a different viewpoint, in the sense that one can try to create models which are basically aimed at creating dispersed solutions. These are called the family of dispersion problems. The classical dispersion models try to maximize dispersion as a function of the distance/dissimilarity among the entities in the network.

The class of dispersion problems is useful when some measure of distance or diversity in the solutions is desirable. For instance in the logistics context it can be used in the location of missile silos where dispersion can reduce the chances of being all attacked or for locating obnoxious facilities to be far from population zones [32]. The dispersion can also be a desirable factor when it comes to franchise location problems where one intends to avoid the cannibalization effects within the chain [76]. The difference is not always translated into the physical distance. For instance, dispersion models can also be used in order to design a portfolio of new products where it is desirable to enter the market with a group of products which are as dissimilar as possible in terms of the quality, price, shape, etc. Another example would be in multiobjective problems where the decision maker might be interested in selecting a collection of solutions as diverse as possible for each objective [72].

Erkut and Neuman [33] propose four different types of the dispersion models based on different dispersion metrics. The first one is called MaxMinSum (Chapter 1) which takes the sum of the distances from each facility to all its neighbors and maximizes the minimum sum of the distances. The second formulation corresponds to the MaxSumSum (Chapter 2) which aims at maximizing the sum of all the hub distances for all located facilities. This model tries to locate p facilities far from a given set of n nodes and far from each other. The third model is the MaxMinMin problem (Chapters 3 and 4) which maximizes the minimum distance between each pair of facilities. And finally the fourth one is the MaxSumMin which seeks to maximize the sum of the minimum distances from each facility to its closest neighbor.

Let V be the set of n vertices, S as any subset of p vertices such that $S \subseteq V, |S| = p$ and $d(i,j)$ the distance between points $i, j \in V$. The four discrete models are formalized as the following general models [33]:

$$
\begin{aligned}
MaxMinSum &: \max\ Z\ s.t.\ Z \leq \textstyle\sum_{j \in S} d(i,j)\ \forall i \in S. \\
MaxSumSum &: \max\ Z\ s.t.\ Z = \textstyle\sum_{i \in S} \sum_{j \in S} d(i,j). \\
MaxMinMin &: \max\ Z\ s.t.\ Z \leq d(i,j)\ \forall i, j \in S, i \neq j. \\
MaxSumMin &: \max\ \textstyle\sum_{i \in S} Z_i\ s.t.\ Z_i \leq d(i,j)\ \forall i, j \in S, i \neq j.
\end{aligned}
$$

The p-dispersion problem is known to be NP-complete by reduction to the clique problem [32]. Ravi et. al [72] also demonstrate that no polynomial time algorithm can guarantee a solution better than twice the optimum for this problem. As the result, the use of heuristics in order to find fast and high quality solutions for large instances or to generate upper bounds seem important [35]. Hansen and Moon [47] also prove that the discrete version of the maximum diversity problem is strongly NP-complete by reduction to the stable set problem.

Exact methods have been developed to solve smaller instances of dispersion problems. Erkut [32] proposed a branch and bound algorithm for the p-dispersion problem. He then proposed a heuristic method to produce better bounds coupled with exact methods in order to solve problems of up to $n = 40$. Agca et al. [2] develop a Lagrangian approach to find optimal solutions for the p-dispersion and maximum diversity problems of size $n = 25$. They

also generate tighter lower and upper bounds for larger instances. More recent combinatorial branch-and-bound methods solve instances of up to $n = 50$ for the maximum diversity problem [3, 62]. Furthermore, the computational experiments in Chapters 1 and 3 of this thesis demonstrate how exact methods are incapable of solving large instances, or even providing tighter bounds for the already obtained solutions by the heuristic methods [76, 78].

Dispersion problems impose a challenge on exact solution procedures and the size of the tackled problems are relatively small for practical implications. Therefore, several metaheuristics and heuristics have been designed to solve much larger instances of various dispersion problems [4, 59, 68]. The state-of-the-art heuristics include variants of `VNS`, hybrid heuristics coupled with other graph theory concepts (maximum clique), Iterated greedy metaheuristic, Iterated tabu search, Learnable tabu search and GRASP [11, 20, 59, 63, 67, 70, 74, 80]. As a matter fact, solving realistically-sized problems that cannot be solved by exact algorithms at all or within a reasonable time, has been one of our motivations to develop elaborate heuristics within the Variable Neighborhood Search (`VNS`) metaheuristic framework.

Variable Neighborhood Search (`VNS`) is a metaheuristic or framework for building heuristics which is based on the idea of a systematic change of the neighborhood in order to escape from the valleys surrounding local optima, followed by a local search to find improved solutions. This general method has been proposed by Mladenović and Hansen [64] and has proven to lead to very successful heuristics for solving large combinatorial programs with applications in location theory, cluster analysis and several other fields. For a recent survey of the theoretical developments and applications including several hundred references see [45, 48].

The main purpose of the proposed methods is to tackle the cannibalization issue in the franchise location literature based on various dispersion criteria which is an innovative approach. The first three chapters study the classical dispersion location models, whereas the fourth chapter focuses on a bi-objective problem including a dispersion and a gravity p-median objective.

The first chapter addresses the MaxMinSum (p-dispersion-sum) problem that consists of the selection of p facilities among n candidate locations in a way that the smallest sum

of the distances among the selected facilities is maximized. This is the first application of any heuristic to this variant of the dispersion problems and the extensive experiments represent various intensification and diversification possibilities within the VNS framework. Comparison with exact methods confirms the high quality of the proposed methods.

The second chapter studies the MaxSumSum (maximum diversity) problem that is relatively well explored by several heuristics in the literature. This problem maximizes the total sum of the distances between each pair of the located facilities. The proposed Greedy VNS heuristics including innovative local search and shake functions improve several results for the known benchmark test problems.

The third chapter addresses the MaxMinMin (p-dispersion) problem, which maximizes the smallest distance among all the selected facilities. The proposed elaborate VNS framework coupled with new plateau search and shake modules, is the first application of VNS to this problem. This method compares favorably to the state-of-the-art heuristics in terms of the quality and robustness of the obtained solutions, lower running times and computational complexity. Besides, several optimal solutions of the largest benchmark instances are found for the first time in the literature.

Finally, the fourth chapter presents an innovative bi-objective model which addresses the dispersion concept among the franchised units as well as the proximity to the customer zones. This is a new model that encompasses the most important factors in real life franchise location practices. In classical operational research facility location models such as the p-center and p-median it is assumed that the clients always prefer to choose the closest facilities in order to receive their desired services. This assumption might be the most appropriate for locating fire stations or hospitals for instance, but is not always the case for locating retail or commercial centers. One of the models trying to explain the clients store patronage behavior is called the gravity rule [1, 15, 25, 53]. Thus, it is important to do modifications in the mathematical formulations in order to create more realistic location models which are consistent with consumer patronage behavior and social sciences studies [5, 10, 13, 21, 54, 55, 56, 60, 75].

Therefore, the last model incorporates the gravity concept in order to do the customer allocation. This is done by the gravity p-median objective proposed by Drezner and Drezner

[24, 25] coupled with the dispersion objective presented by the MaxMinMin criterion. The proposed VNS for this bi-objective problem is also its first application to the gravity p-median problem. In the gravity p-median problem customers are not necessarily allocated to the closest facility, and that are allocated proportional to the attractiveness of that facility and to a decreasing utility function of the distance to the facility. Experiments on known p-median datasets explore the best VNS setting in order to approximate the Pareto front. Besides, the proposed method finds the best known solutions for the gravity p-median problem in a very short computational time.

In the following we intend to present a more integrated view of the proposed VNS-based heuristics for the above three dispersion problems, i.e., MaxMinSum (Chapter 1), MaxSum-Sum (Chapter 2) and MaxMinMin (Chapters 3 and 4). Therefore, we designed a series of complementary tests in the General Conclusion section in order to highlight the similarities and the differences among the suggested methods in terms of both their *technical* VNS *configurations* and obtained *results* for the large datasets.

In order to better understand the *technical similarity* of the proposed methods for the above dispersion problems, we first refer to the classical structure of VNS. The general VNS framework includes three main modules: 1) Initialization method, 2) Shake function and 3) Local Search procedure. The initialization and Shake methods could be done in random or various greedy manners and the Local Search function incorporates a Contribution calculation function and a parameters Update procedure. The main idea regarding the Contribution and Update procedures is to store intelligently some problem specific parameters that would allow for efficient calculations with low computational complexity. *All* the proposed methods use the above main modules and ideas, of course with different calculation methods and algorithmic procedures based on the very characteristics of each dispersion problem. An important concept shaping the general VNS framework is the diversification versus intensification factor. At each iteration of VNS either an improvement is made or not. In case of no improvement a decision on how to start the next iteration should be made. The next iteration is either started form the already best solution obtained, or from the current solution just obtained. The former will lead to more intensification in the search, whereas the latter favors diversification. Besides, the Local Search procedure can pursue a first improvement strategy favoring more diversification, versus best improvement strategy

leading to more intensification. Another technical similarity of *all* the methods (supported by empirical and preliminary test results) is that they pursue the *from current* `VNS` strategy and the *first contribution* Local Search.

One major *technical difference* of the proposed methods is caused by the objective function structure of the dispersion problems. Due to the first "Min" operator of the MaxMinSum and MaxMinMin problems, only one binding value would define the objective function value. This would complicate the Contribution and Update solution procedure, and thus more detailed and elaborate methods should be designed in this respect.

Next, the same comparisons are done in terms of the obtained *results*. For all the three dispersion problems it is concluded that the Random initialization method should be used in order to find new improved solutions in repeated runs of the tests, while Greedy starting methods should be used if longer running times are allowed. The final tests explained later in the General Conclusion section confirm that the use of intelligent shake functions would also significantly improve the quality of the obtained solutions for *all* the dispersion problems. This is due to the fact that the purely random modules cannot find high quality results (within a close percentage of the best known solutions) for the larger instances. Another observation is that apart from the instance sizes, the data structure of the instances plays also an important role in terms of the difficulty of the solution procedure. Although we have not found any consistent empirical rule between the p/n ratio and the number of obtained best known solutions, the results demonstrate that smaller p/n ratios might be more challenging to solve.

The main difference of the *results* for various dispersion problems lies in the minor tuning of the parameters such as the maximum shake size, etc. Besides, the MaxMinSum and MaxMinMin problems obtain better results with the SemiGreedy shake function, whereas the MaxSumSum problem performs better with the Greedy Shake function. Another factor tested in order to improve the quality of the solutions is allowing longer running times. The results show that this would improve the quality of the solutions, yet not within the small 0.5% distance of the best known solutions. It should be noted that the exact methods are not capable of finding results within a reasonable percentage of the best known solutions

which confirms one more time the need for heuristic solution procedures to tackle large and challenging dispersion problem instances.

The contribution and novelty of each method is explained in details in the following chapters followed by the General Conclusion of the thesis. The following four chapters are designed as independent research papers and can be read individually. As the result, there may be some repetitions in the content or slight differences in the notation from one chapter to another.

Chapter 1

Variable Neighborhood Search Heuristic Methods for the MaxMinSum (*p-Dispersion-Sum*) Problem

Behnaz Saboonchi

Pierre Hansen

Sylvain Perron

Department of Management Sciences

GERAD and HEC Montréal

3000, chemin de la Côte-Sainte-Catherine

Montréal, Québec, H3T 2A7, Canada

Abstract

Dispersion problems consist of the selection of a fixed number of vertices from a given set so that some function of the distances among the vertices is maximized. Such problems impose a challenge on heuristic and metaheuristic solution procedures. Among different variations of the dispersion models, the MaxMinMin (p-dispersion) and the MaxSumSum (maximum diversity) problems have been the subject of much research, yet the MaxMinSum problem has not been well explored in the literature. In this paper we have developed several heuristics based on the Variable Neighborhood Search metaheuristic framework, including various greedy constructive procedures and different shaking strategies. Finally we discuss the tradeoffs among different solution strategies and compare our results with those of exact methods for smaller-sized instances which confirm the high quality of our solutions. To the best of our knowledge this is the first application of any heuristic for the MaxMinSum dispersion problem and the results of our extensive computational experiments on large datasets would set a new benchmark for future comparison purposes.

1.1 Introduction

In the family of dispersion problems, given a set of n vertices we intend to select a subset of size p in a way that some function among the selected vertices is maximized. This is useful when some measure of distance or diversity in the solutions is desirable. For instance in the logistics context it can be used in the location of missile silos where dispersion can reduce the chances of being all attacked or for locating obnoxious facilities to be far from population zones [32]. The dispersion can also be a desirable factor when it comes to franchise location problems where one intends to avoid the cannibalization effects within the chain. The difference is not always translated into the physical distance. For instance, dispersion models can also be used in order to design a portfolio of new products where it is desirable to enter the market with a group of products which are as dissimilar as possible in terms of the quality, price, shape, etc. Another example would be in multi-objective problems where the decision maker may be interested in selecting a collection of solutions as far as possible for each objective [72].

Erkut and Neuman [33] propose four different types of the dispersion models based on different dispersion metrics. The first one is the MaxMinMin problem which maximizes the minimum distance between each pair of facilities. The second one is the MaxSumMin which seeks to maximize the sum of the minimum distances from each facility to its closest neighbor. The third formulation is called MaxMinSum which takes the sum of the distances from each facility to all its neighbors and maximizes the smallest sum of the distances. Finally, the fourth formulation corresponds to the MaxSumSum which aims at maximizing the sum of all the hub distances for all the located facilities. This model tries to locate p facilities far from a given set of nodes and far from each other.

The idea of permuting the operators *sum* and *max* in order to create new location problems has also been used for other well known problems such as the p-median and p-center [65, 43]. The classical p-median problem has a p-sum-sum objective function (i.e., the sum over p facilities of the sum of the distances to the clients assigned to it), and the classical p-center problems has a p-max-max objective function (i.e., the maximum over p facilities of the maximum distance to each client assigned to it). Hansen et al. [43] introduce two new variants of such problems and discuss their real life applications.

As mentioned by Curtin and Church [18, 19], the MaxMinSum dispersion problem was first introduced in the location literature with the review of the dispersion objective metrics by Erkut and Neuman [33]. They use this problem as part of their family of multiple-type dispersion formulations [18]. They consider the distances included in the objective function as a hub distance where each facility located at location i is at the hub of a wheel, and the spokes of the wheel radiate out from i to all other located facilities at locations j.

The MaxMinSum problem is similar to the MaxSumSum (maximum diversity) problem in the sense that they both share the concept of the *sum* measure for the distances, yet in the latter the sum of the distances for all the selected locations is considered in the objective function. On the other hand it is also similar to the MaxMinMin (p-dispersion) problem which aims at maximizing the smallest closest distance between any selected location and the other selected ones. Both objectives try to optimize the worst-case performance, yet in the former the *sum* of the distances from each selected location to all others is used in the objective function.

The MaxMinSum and MaxSumSum problems are known to be good replacements for each other for smaller-sized instances [33]. Therefore, the developed heuristics in this work could also be used as reasonable substitute methods for large and challenging MaxSumSum instances.

Due to this similarity, throughout this paper we call the MaxMinSum problem as the "*p-dispersion-sum problem*". This problem was first modeled by Erkut and Neuman [33] as the following mixed 0-1 linear program:

$$\begin{aligned} \max \quad & Z \\ \text{s. t} \quad & Z \leq \sum_{i=1}^{n} d(v_i, v_j) x_i + M(1 - x_j) \quad 1 \leq j \leq n \\ & \sum_{i=1}^{n} x_i = p \\ & x_i = \{0, 1\} \qquad\qquad\qquad\qquad 1 \leq i \leq n, \end{aligned}$$

where x_i is a binary decision variable defining if vertex v_i is selected and $d(v_i, v_j)$ is the distance between any pair of the located facilities at locations i and j. The distances between all the vertices are taken as an input and stored in an $n \times n$ upper-triangular matrix with $d(v_i, v_i) = 0$. It should be noted that M is a sufficiently large value which could be set as $p \times d_{max}$, where d_{max} is the largest distance between any pair of locations. In Section 1.3.3 an upper bounding technique in order to obtain tighter bounds is discussed.

In Section 1.2 we present a detailed explanation of our proposed `VNS` heuristic solution procedure for the p-dispersion-sum problem. Then we discuss our computational experiments on benchmark test problems coupled with comparison with exact methods in Section 1.3. Finally we conclude our paper by highlighting our contributions and suggestions for future research.

1.2 VNS for The *p*-Dispersion-Sum Problem

Variable Neighborhood Search (`VNS`) is a metaheuristic or framework for building heuristics which is based on the idea of a systematic change of the neighborhood in order to escape from the valleys surrounding local optima, followed by a local search to find improved solutions. This general method has been proposed by Mladenović and Hansen [64] and has

proven to lead to very successful heuristics for solving large combinatorial programs with applications in location theory, cluster analysis and several other fields. For recent surveys see e.g. [45, 48].

Within the family of dispersion problems the `VNS` method has been applied to the maximum diversity and p-dispersion problems and have beeb proven to be among the most efficient methods compared to other heuristics [11, 63, 76, 77]. Therefore, we have decided to develop a heuristic method within the VNS framework that is well-suited to the p-dispersion-sum problem.

We first express the p-dispersion-sum problem in graph theoretical terms. Let $V = \{v_i, \forall i = 1, \ldots, n\}$, be a set of n vertices (potential locations) and v_i representing each member of this set. Let E be the set of $\binom{n}{2}$ edges of an undirected fully-connected graph $G(V, E)$, with $d_e > 0$ representing the distance over each edge $e \in E$. The value p is an integer such that $3 \leq p \leq |V|$. We define S as any subset of p vertices such that $S \subseteq V, |S| = p$. The subset of the vertices not present in the current solution is defined as $\bar{S}$ such that $\bar{S} = V \setminus S, |\bar{S}| = n - p$.

The objective function value $f(S)$ at each step is defined as the smallest sum of the distances between each selected vertex and the rest of the selected vertices induced by the subset S:

$$f(S) = \min_{v_i \in S} \left\{ \sum_{v_j \in S} d(v_i, v_j) \right\}.$$

The p-dispersion-sum problem intends to find the optimal subgraph $G(S^*, E(S^*))$, where:

$$S^* = \arg\max_S f(S).$$

The solution space U is represented by the $\binom{n}{p}$ subsets of V with cardinality p. In order to apply `VNS`, a metric function is defined to evaluate the distance between any two solutions S and S':

$$\delta(S, S') = \delta(S', S) = |S \setminus S'|.$$

Based on the metric distance function defined above, the neighborhood of size k of a solution S is defined as:

$$N_k(S) = \{S' \in U | \delta(S, S') = k\}; k = 1, 2, \ldots, \min\{p, n - p\}.$$

In order to represent the solution at each step of the heuristic we use the data structure suggested by Brimberg et al. [11]. The solution is represented by an array of the n indices corresponding to each vertex or candidate location, where the first p elements correspond to the subset of the current solution S.

Throughout this paper the following notations are used:

- $f(S_{best/cur})$: the best/current objective function value that corresponds to the smallest sum of the distances for each vertex in the best/current solution set S;
- $W(v_i)$: the sum of the distances from any vertex v_i $(i = 1, \ldots, n)$ to all the vertices in the solution set S;
- v_{exit}: the vertex inside the solution set that is a candidate to *leave* the solution set $(v_{exit} \in S)$;
- v_{enter}: the vertex outside the solution set that is a candidate to *enter* the solution set $(v_{enter} \in \bar{S})$.

The above values are first computed at the construction of the initial solution and are updated each time a new solution is found.

Algorithm 1.1 presents the `VNS` function and then in the following sections we explain in details the functions embedded in the general framework. The stopping criterion is the total execution time t_{max} and the already elapsed cumulative time in the overall procedure is noted by $t_{elapsed}$. The k_{min} and k_{step} (shake step size) parameters are set by default to 1, and the k_{max} (maximum shake size) is set to $\min\{p, n - p\}$.

```
function VNS (k_min, k_step, k_max)
S_cur ← Initialize();
S_best ← S_cur;
t_elapsed = 0;
k_max = min{p, n − p};
while t_elapsed ≤ t_max do
    k_cur ← k_min;
    while k_cur ≤ k_max and t_elapsed ≤ t_max do
        S_cur ← Shake(S_cur);
        S_cur ← LocalSearch(S_cur);
        if f(S_cur) > f(S_best) then
            S_best ← S_cur;
            k_cur ← k_min;
        else
            k_cur ← k_cur + k_step;
        end
    end
end
```

Figure 1.1: Pseudo Code for the VNS Framework

1.2.1 Initialization

The initial solution could be created at random or in a greedy manner. Based on the *random* method the initial solution is simply created by choosing p indices at random.

Two *Greedy* construction heuristics have been widely used in the literature in order to create initial solutions for the dispersion problems [35]. The *Greedy deletion* heuristic starts with all the n vertices and eliminates one vertex at each iteration. The deletion candidate is the one with the smallest sum of the distances to the rest of the remaining vertices at each iteration, and the ties are broken arbitrarily. Of course this procedure is repeated $(n - p)$ times until exactly p vertices remain in the solution set.

The *Greedy add* heuristic selects a starting vertex at random and creates the complete solution set in $(p-1)$ iterations by adding the vertex with the largest increase in the objective value [4, 11, 59]. As the result the size of the under construction solution set is smaller than p and will gradually reach the complete size as the construction phase is terminated. The under construction solution set could be represented as C, and the set of the vertices outside this set as $\bar{C}$. After the addition of the entering vertex $v_{enter} \in \bar{C}$, the existing sum of the

distances values will be updated as: $W(v_i) + d(v_i, v_{enter})$ for all the $v_i \in C$. As the result the objective function value after the addition of each v_{enter} will be the minimum value among the updated $W(v_i)$ (for all the $v_i \in C$), plus the newly added $W(v_{enter})$ value.

This heuristic could be repeated n times based on different starting vertices and then the best one leading to the highest objective value could be selected. Based on preliminary results we observed that this heuristic is very time consuming (much more than the *Greedy deletion* heuristic), and for large datasets it is not worthwhile to take this procedure just to further improve the initial solution. In order to overcome this drawback the *Greedy add* heuristic is initialized by choosing the two furthest vertices as the initial vertices and by repeating the above-mentioned procedure $(p-2)$ times. We know empirically that the results obtained by this method is among the highest possibilities for the *Greedy add* without spending too much computational time on the initial solution.

1.2.2 Local Search

After having created the initial solution, the `LocalSearch` procedure is implemented performing 1-interchange swaps on the current solution as shown in Algorithm 1.2. This means that at each iteration only *one* vertex is swapped at a time. The swap could be done whenever the first (first improvement strategy) or the best (best improvement strategy) contribution is made to the current objective value.

In order to start the `LocalSearch` procedure the gain obtained from swapping the selected entering candidate with the selected leaving candidate should be evaluated. The two main `Contribution` and `Update` functions will be explained in details in the following.

```
function LocalSearch(S)
repeat
    (v_exit, v_enter, gain) ← Contribution(S);
    if gain > 0 then
        Swap(v_exit, v_enter);
        Update(v_exit, v_enter, S);
    end
until gain > 0;
```

Figure 1.2: Pseudo Code for the Local Search Procedure

1.2.2.1 Contribution

Algorithm 1.3 presents the `Contribution` function. In the proposed `LocalSearch` procedure for each leaving vertex $v_{exit} \in S$ chosen randomly, the `Contribution` function can determine the first or best entering candidate $v_{enter} \in \bar{S}$, as well as its corresponding contribution to the current objective function value.

function `Contribution`(S)
$gain = 0$;
choose a random v_{exit} and v_{enter};
$i = 0; j = p$;
while $i \leq p$ *and* $gain = 0$ **do**
  while $j \leq n - p$ *and* $gain = 0$ **do**
    $f(S_{temp}) = f(S_{cur})$;
    if $W(v_{enter}) - d(v_{exit}, v_{enter}) > f(S_{cur})$ **then**
      forall the $(v_i \in S; i \neq exit)$ **do**
        $W(v_i) \leftarrow W(v_i) - d(v_i, v_{exit}) + d(v_i, v_{enter})$;
        remove $W(v_{exit})$ from $f(S_{temp})$;
        add $W(v_{enter}) - d(v_{exit}, v_{enter})$ to $f(S_{temp})$;
        $f(S_{temp}) \leftarrow \min\{W(v_i)\}$;
      end
      if $f(S_{temp}) > f(S_{cur})$ **then**
        gain $\leftarrow f(S_{temp}) - f(S_{cur})$;
      else
        $j++$;
      end
    else
      $j++$;
    end
  end
  $i++$;
end
if $gain > 0$ **then**
  return $(v_{exit}, v_{enter}, \text{gain})$;
end

Figure 1.3: Pseudo Code for Determining the First Improving Swap and its Contribution (First Improvement Strategy)

Unlike the MaxSumSum (maximum diversity) problem, the evaluation of the change in the objective function value associated with each swap is not straightforward. The reason is that in the MaxSumSum problem the overall sum of all the distances among the vertices in the solution set should be evaluated, whereas in the p-dispersion-sum problem the smallest

sum of the distances (the worst case scenario) should be improved in each `LocalSearch` iteration.

In order to initiate the `Contribution` function a random leaving candidate $v_{exit} \in S$, and a random entering candidate $v_{enter} \in \bar{S}$ are chosen. Before evaluating the possible gain derived from the swap, a preliminary check is done in order to verify if the selected random entering candidate v_{enter} would definitely deteriorate the current solution. In order to do so the updated $W(v_{enter})$ is calculated which corresponds to the sum of the distances value for the entering candidate in case it enters the solution set. This value is updated by: $W(v_{enter}) - d(v_{exit}, v_{enter})$ and will be added to the objective function, so if it's already inferior than the current objective function value (the already smallest sum of the distances) the new solution after the swap will definitely be worse. As the result such a swap will be abandoned and the algorithm moves on to the next entering candidate.

In order to calculate the possible gain after the swap the updated sum of the distances values for the vertices already in the solution set (for all the $v_i \in S$) are required by calculating $W(v_i) - d(v_i, v_{exit}) + d(v_i, v_{enter})$. Then the $W(v_{exit})$ is removed from, and the $W(v_{enter})$ is added to the objective function. The smallest updated sum of the distances $W(v_i)$ for all $v_i \in S$ after the possible swap will determine the new objective function value. This value is calculated in $O(p)$ time and if it's better than the current solution the swap will be accepted, if not the algorithm will proceed to the next entering candidate until it finds an improvement.

With the first improvement strategy the `Contribution` function stops as soon as an improving solution is found, as a result in the worst case it is implemented in $O(p(n-p)p) = O(np^2)$ time per `LocalSearch` iteration.

1.2.2.2 Update

The `Update` procedure performs all the required updates before proceeding to the subsequent iteration. It is implemented in two different phases in order to update all the $W(v_i)$ sum of the distances, and then to update the $f(S_{cur})$ which represents the objective function value for the current solution.

The update for the $W(v_i)$ is straightforward and is performed in $O(n)$ total time. As already mentioned the chosen leaving candidate is represented by v_{exit} and the entering candidate by v_{enter}, thus the updated $W(v_i)$ would be:

$$W(v_i) = W(v_i) + d(v_i, v_{enter}) - d(v_i, v_{exit}), i = 1, 2, \ldots, n.$$

The update of the objective function is done in $O(p)$ time at each iteration by finding the minimum sum of the distances for the vertices in S after having updated the respective $W(v_i)$ values.

1.2.3 Shake

The perturbation in most `VNS`-based heuristics is done in a simple manner by selecting a random vertex from the k_{th} neighborhood, i.e. $N_k(S)$ from the current solution S and then repeating k times the random swap move. The `RandomShake` function does so by choosing one random leaving and entering candidate at each iteration with updates in between each swap. However, in this paper we have developed two additional shake functions in order to control the perturbation operation in a more intelligent manner.

The `SemiGreedyShake` function fixes a random leaving candidate from the current solution set S and then chooses an entering candidate that has the highest sum of the distances value in case it replaces the exit candidate, i.e. the highest $W(v_{enter}) - d(v_{exit}, v_{enter})$ value. This method does not guarantee that the selected candidate is the best among all the insertion candidates. This is due to the fact that in order to find the best swap (after having fixed the exit candidate) leading to the lowest deterioration or maybe highest gain in the objective function value, one needs to verify also the updated $W(v_i)$ values for all the $v_i \in S$ in case the entering candidate is added to the solution set, and then to repeat the same procedure to calculate the overall objective function value for all the possible entering candidates in order to select the best one. This approach would be the same as the procedure already explained in the `Contribution` function. Yet, we decide not do to so as the first method is performed in $O(n - p)$ time, whereas the second one is done in $O((n - p)p)$. Therefore, this method adds more randomness to the shake function rather than proceeding as the `Contribution`

function and choose the best swap. This procedure is repeated until the shake size of k is attained. Each iteration is performed in $O(n-p)$ time and after each swap the `Update` function is called.

In order to have a more intensified shake operation we have developed the `GreedyShake` function which for a shake of size k, selects the k vertices with the smallest $W(v_i)$ values for all $v_i \in S$, and swaps all of them with the k vertices with the largest $W(v_i)$ values for all $v_i \in \bar{S}$. This greedy fashion of selecting the entering candidates could provide better starting vertices for the subsequent `LocalSearch` procedure. The k leaving and entering candidates are chosen all at once and are not changed while the updates are performed between the swaps. The performance of the two shake functions will be compared in details in Section 1.3 where it is demonstrated that the more intelligent shake functions dominate the purely random ones.

1.3 Computational Experiments

In this section we have selected four of the largest benchmark instances in the maximum diversity problem literature that were collected by Martí et al. [63], leading to 80 instances in total. A brief description of the characteristics of the datasets is given below:

- SOM-b: this dataset consists of 20 matrices each with random numbers between 0 and 9 generated from an integer uniform distribution by Silva et al. [79]. The instance sizes are such that for $n = 100$, $p = 10$, 20, 30 and 40; for n= 200, $p = 20$, 40, 60 and 80; for $n = 300$, $p = 30$, 60, 90 and 120; for $n = 400$, $p = 40$, 80, 120, and 160; and for $n = 500$, $p = 50$, 100, 150 and 200.

- MDG-a, MDG-b: these datasets consist of 20 matrices each with real numbers randomly selected between 0 and 10 from a uniform distribution by Duarte and Martí [27] with $n = 2000$ and $p = 200$.

- MDG-c: this dataset consists of 20 matrices with $n = 3000$ and $p = 300$, 400, 500 and 600. The MDG instances have been used in [67].

First we describe our experiments that were designed to study the performance of different settings within the `VNS` framework and then compare and analyze the tradeoffs and overall

results obtained by different methods over all the test problems. The results are finally compared with that of exact methods for the relatively smaller instances.

All the heuristics were coded in C ++ and run on a linux machine with an intel processor of 2.667 GHz and 3Gb of Ram. The best obtained results after two hours of running time are reported as suggested in [63].

1.3.1 Experiments Setup

As mentioned in Section 1.2 the proposed `VNS` implementation allows various settings and methods within its framework. In order to initialize the `VNS` three different methods have been discussed: Random add (RA), Greedy add (GA) and Greedy deletion (GD). There are also three different shaking possibilities: Random shake (RS), Semi-Greedy shake (SG) and Greedy shake (GS). In the general framework presented in Section 1.2 the shake size at each iteration increases systematically and will be reset to k_{min} whenever an improvement is made or when the k_{max} value is reached. The k_{max} is a parameter whose value by default is $\min\{p, n-p\}$, which could be a large value depending on the problem size. Therefore, a smaller value, i.e. $(0.75 * \min\{p, n-p\})$ is used to verify if it helps improve the performance of the heuristic. This will lead to $3 \times 3 \times 2 = 18$ combinations of different `VNS` modules.

On the other hand at each iteration of `VNS` either an improvement is made or not. In case of no improvement a decision on how to start the next iteration should be made. The next iteration is either started form the already best solution obtained (from best or FB), or from the current solution just obtained (from current or FC). The former will lead to more intensification in the search, whereas the latter favors diversification. Besides, the `LocalSearch` procedure can pursue a first improvement strategy ($FirstI$) favoring more diversification, versus best improvement strategy ($BestI$) leading to more intensification. The above mentioned intensification and diversification strategies will lead to four general `VNS` frameworks. As the result the datasets are run under the $18 \times 4 = 72$ total combinations.

We do not allow longer running times in order to get further improvements, as a result the tests are run *only once* under *two hours* of running time. Throughout the paper we present the average % *deviation* from the best known solutions obtained by the 72 combinations for each group of dataset:

$$\% \text{ deviation} = \frac{\text{best value - actual value}}{\text{best value}} \times 100.$$

The best known solutions for all the 80 data instances are presented in Table 1.IV . The average deviations presented in Tables 1.I to 1.III are all calculated based on the results presented in the fourth column (All methods) of Table 1.IV which refer to the best solutions obtained by the 72 combinations.

Table 1.I represents the average deviation from the best solutions obtained for all the 80 data instances for each of the *four* `VNS` general frameworks, *three* initialization methods, *three* shaking strategies and the *two* possible k_{max} sizes. The smallest average deviation values for each group are shown in bold under the *Average* column. Same representation has also been done to highlight the smallest average deviation values for each dataset separately.

The first part of the table which refers to the four general `VNS` frameworks reveals that the current iteration strategy and first contribution local search *(FC-FirstI)* leads to the lowest average deviation. This is also true for each individual dataset except for the SOM-b instances where the *(FB-FirstI)* strategy leads to the lowest average deviation. In the second part which refers to the problem initialization methods, the Greedy deletion strategy (*GD*) consistently leads to the lowest average deviation for all datasets. In the third group which addresses the shaking strategy it is clearly observed that the Semi-Greedy shake (*SG*) strategy has a significantly lower average deviation across all the datasets.The only parameter in the experiments is the maximum shake size which is presented in the last comparison group. As it is seen the smaller maximum shake size is slightly better for smaller-sized instances, and the bigger maximum shake size leads to slightly lower average deviations for the largest dataset. The maximum shake size is a parameter that can be tuned in order to obtain better results. Here we are more interested in the main modules of `VNS` rather than tuning its parameters and since it is observed that the maximum shake size does not make a large difference on the average deviations, we decide not to further experiment on this factor.

The above mentioned results for individual factors and settings do not necessarily lead to overall better combinations. As the result in Table 1.II we present the same results as in Table 1.I in a different manner by calculating the average deviations of the 18 combinations

Table 1.I : Average Deviation % for Individual Methods

	SOM-b	MDG-a	MDG-b	MDG-c	Average
General framework					
FC-FirstI	2.58	**2.55**	**2.25**	**1.45**	**2.21**
FC-BestI	3.02	2.9	2.55	1.72	2.55
FB-FirstI	**2.02**	2.83	2.38	1.77	2.25
FB-BestI	3.6	3.52	2.98	2.13	3.06
Initialization					
RA	4.58	5.73	5.5	3.72	4.88
GA	2.4	1.96	2.01	1.11	1.87
GD	**1.44**	**1.16**	**0.99**	**0.47**	**1.02**
Shake method					
RS	5.58	6.02	5.28	3.72	5.15
SG	**0.59**	**0.72**	**0.85**	**0.38**	**0.64**
GS	2.25	2.1	2.37	1.21	1.98
Shake max					
0.75p	**2.75**	**2.94**	**2.82**	1.77	**2.57**
p	2.86	2.96	2.85	**1.76**	2.61

based on the four different frameworks for all the test problems. On the last row the overall average deviation for the four `VNS` frameworks has been calculated and on the last column the same has been done for the 18 settings. It is clearly observed that the current iteration strategy and first contribution local search *(FC-FirstI)* leads to the lowest average deviation over all the datasets and settings followed by the *(FB-FirstI)*. This shows the superior performance of the first improvement strategy which has already been observed in the literature [11, 44]. Finally the three lowest average deviations belong to *(GD-SG-p)*, *(GD-SG-0.75p)* and *(RA-SG-p)* settings in increasing order.

So far all the comparisons made were based on the average deviation from the best ever solutions which refer to the average quality of the solutions derived from each method. Yet, another important indicator of a good `VNS` setting is the number of times it obtains the best known solutions. As the result we prepared Table 1.III which represents the number of best known solutions obtained over all the 80 data instances, for each of the combinations.

From Tables 1.II and 1.III three important observations can be highlighted:

- The current iteration strategy and first contribution local search *(FC-FirstI)* leads to the lowest average deviations and higher number of found best known solutions compared to other `VNS` frameworks.

Table 1.II : Average % Deviation for All Combinations and Frameworks

Method	FC-FirstI	FC-BestI	FB-FirstI	FB-BestI	Average
RA-GS-0.75p	1.32	1.1	2.55	4.72	2.42
RA-GS-p	1.57	1.31	2.56	4.95	2.6
RA-SG-0.75p	0.13	0.36	0.85	1.13	0.62
RA-SG-p	0.13	0.41	0.77	0.99	**0.57**
RA-RS-0.75p	10.75	13.73	8.49	11.26	11.06
RA-RS-p	10.85	13.75	8.65	11.61	11.21
GA-GS-0.75p	1.32	1.03	1.99	2.89	1.81
GA-GS-p	1.63	1.3	2.04	2.89	1.96
GA-SG-0.75p	0.13	0.42	1.23	1.24	0.76
GA-SG-p	0.12	0.44	0.98	1.11	0.66
GA-RS-0.75p	2.93	3	2.53	2.69	2.79
GA-RS-p	2.95	3	2.5	2.73	2.79
GD-GS-0.75p	1.13	0.96	0.93	1.45	1.12
GD-GS-p	1.22	1.02	0.93	1.45	1.15
GD-SG-0.75p	0.1	0.27	0.55	0.6	**0.38**
GD-SG-p	0.12	0.26	0.48	0.51	**0.34**
GD-RS-0.75p	1.67	1.74	1.21	1.39	1.5
GD-RS-p	1.71	1.74	1.28	1.4	1.53
Average	**2.21**	2.55	2.25	3.06	

Table 1.III : Number of Best Solutions Obtained for All Combinations and Frameworks

Method	FC-FirstI	FC-BestI	FB-FirstI	FB-BestI	Average
RA-GS-0.75p	3	4	0	0	7
RA-GS-p	0	3	1	0	4
RA-SG-0.75p	24	7	0	1	32
RA-SG-p	28	5	1	1	**35**
RA-RS-0.75p	1	0	1	0	2
RA-RS-p	0	0	0	0	0
GA-GS-0.75p	4	4	1	0	9
GA-GS-p	2	2	1	0	5
GA-SG-0.75p	21	6	1	1	29
GA-SG-p	23	5	1	1	30
GA-RS-0.75p	1	0	0	0	1
GA-RS-p	0	0	0	0	0
GD-GS-0.75p	2	5	2	1	10
GD-GS-p	2	4	1	1	8
GD-SG-0.75p	27	11	2	2	**42**
GD-SG-p	20	10	2	2	**34**
GD-RS-0.75p	1	0	1	0	2
GD-RS-p	0	0	0	0	0
Sum	**159**	66	15	10	

- The three best methods averaged over all the four `VNS` frameworks are *(RA-SG-p),(GD-SG-0.75p)* and *(GD-SG-p)*.

1.3.2 Post-Hoc Analysis

The above mentioned *(GD-SG-0.75p)*, *(RA-SG-p)* and *(GD-SG-p)* methods under the current iteration strategy and first contribution local search *(FC-FirstI)* `VNS` framework are the most promising methods based on both the average deviation and also number of best solutions obtained criteria. As mentioned before we did not allow longer running times and the heuristics were run only once. As the result two questions seem interesting: 1) Do multiple runs of the heuristics lead to new improved solutions? and 2) Do longer running times improve the quality of the solutions substantially? In real life applications the answer to these questions become more important when computational resources are of great concern for the decision makers or if faster solutions with relatively higher qualities are preferred rather than taking chances in hope of new improved solutions.

It should be noted that all the three above-mentioned heuristics use the Semi-Greedy shake (*SG*) method, yet two of them start from a Greedy deletion (*GD*) initialization and the other one from a Random start (*RA*) initial solution. Here we would like to answer the above two questions taking the initialization and shaking methods into account, and as already discussed the maximum shake size is a parameter that could be further improved empirically which is not of our interest in this work.

The first question addresses the robustness of the heuristics. We expect that the methods with the greedy starts to be more robust than the random start methods due to their more intensified approach, whereas for the random start method we expect higher chances of new improved solutions in repeated runs. In order to verify this idea the three *(GD-SG-0.75p)*, *(RA-SG-p)* and *(GD-SG-p)* heuristics are run under the current iteration strategy and first contribution local search *(FC-FirstI)* `VNS` framework, 30 times with the same two hours of running time, only for the MDG-c group as it's the largest dataset. As expected, the *(GD-SG-0.75p)* and *(GD-SG-p)* methods obtain the same results as before for all the 30 runs, whereas the *(RA-SG-p)* heuristic results in different solutions in different runs. What's more interesting is that the random start method leads to several new improvements. Based on the results obtained on the MDG-c largest dataset we conclude that more chances of improved solutions can be expected only by multiple runs of the *random start* heuristic. As

the result for the other datasets only the *(RA-SG-p)* heuristic is used for the multiple run experiment and the greedy start ones are no longer tested.

The best results over the 30 runs of the *(RA-SG-p)* heuristic are presented for each dataset in the fifth column (Multiple runs) of Table 1.IV . Comparison of the fourth and the fifth columns of Table 1.IV reveals that the multiple (30) runs experiment has led to 10, 15 and 6 new improvements for the MDG-a, MDG-b and MDG-c datasets respectively. The reported values are the best among 30 runs, yet in order to have a better idea of the overall quality of all the 30 solutions, the coefficient of variation *(CV)* is presented for each dataset within parentheses in the fifth column of Table 1.IV :

$$\text{CV} = \frac{\text{standard deviation}}{\text{mean}} \times 100.$$

As it is seen for all the 80 instances this value is very small, much less than 1%, which shows that our random methods are also very robust.

Table 1.IV : Best Known Solutions for All the Datasets

Instance	n	p	All methods	Multiple runs	Long run
SOM-b-1	100	10	62	62 (0)	62
SOM-b-2	100	20	111	111 (0)	111
SOM-b-3	100	30	151	151 (0)	151
SOM-b-4	100	40	195	194 (0)	194
SOM-b-5	200	20	117	117 (0)	117
SOM-b-6	200	40	212	212 (0)	212
SOM-b-7	200	60	298	298 (0)	298
SOM-b-8	200	80	386	386 (0.13)	386
SOM-b-9	300	30	170	170 (0.29)	169
SOM-b-10	300	60	309	309 (0.11)	308
SOM-b-11	300	90	440	440 (0.09)	440
SOM-b-12	300	120	572	572 (0)	572
SOM-b-13	400	40	222	222 (0)	222
SOM-b-14	400	80	405	405 (0)	405
SOM-b-15	400	120	580	579 (0.07)	579
SOM-b-16	400	160	752	752 (0)	752
SOM-b-17	500	50	272	272 (0.19)	272
SOM-b-18	500	100	503	503 (0.09)	503
SOM-b-19	500	150	726	724 (0.04)	724

Table 1.IV : Best Known Solutions for All the Datasets (Continued)

Instance	n	p	All methods	Multiple runs	Long run
SOM-b-20	500	200	937	937 (0.05)	937
Average			**371**	370.8	370.7
MDG-a-21	2000	200	1100	1101 (0.1)	1099
MDG-a-22	2000	200	1101	1101 (0.11)	1101
MDG-a-23	2000	200	1102	1102 (0.13)	1102
MDG-a-24	2000	200	1100	1099 (0.08)	1101
MDG-a-25	2000	200	1100	1101 (0.15)	1099
MDG-a-26	2000	200	1103	1101 (0.07)	1099
MDG-a-27	2000	200	1103	1104 (0.29)	1105
MDG-a-28	2000	200	1101	1101 (0.12)	1101
MDG-a-29	2000	200	1100	1101 (0.12)	1101
MDG-a-30	2000	200	1101	1101 (0.1)	1101
MDG-a-31	2000	200	1100	1102 (0.14)	1098
MDG-a-32	2000	200	1101	1099 (0.05)	1099
MDG-a-33	2000	200	1101	1101 (0.11)	1099
MDG-a-34	2000	200	1102	1100 (0.09)	1101
MDG-a-35	2000	200	1101	1102 (0.13)	1100
MDG-a-36	2000	200	1101	1103 (0.12)	1101
MDG-a-37	2000	200	1101	1102 (0.12)	1100
MDG-a-38	2000	200	1104	1104 (0.11)	1104
MDG-a-39	2000	200	1100	1101 (0.12)	1101
MDG-a-40	2000	200	1102	1103 (0.09)	1102
Average			1101.2	**1101.45**	1100.7
MDG-b-21	2000	200	109187.6	109300.15 (0.09)	109187.6
MDG-b-22	2000	200	108941.82	109021.67 (0.05)	108925.07
MDG-b-23	2000	200	109017.82	109437.88 (0.09)	109075.22
MDG-b-24	2000	200	108989.58	108990.9 (0.06)	109131.75
MDG-b-25	2000	200	109191.14	109188.59 (0.09)	109023.6
MDG-b-26	2000	200	109077.85	109190.19 (0.07)	109131.75
MDG-b-27	2000	200	109116.41	109135.82 (0.06)	109154.95
MDG-b-28	2000	200	108951.47	109063.15 (0.07)	109015.79
MDG-b-29	2000	200	109173.85	109137.43 (0.06)	109185.2
MDG-b-30	2000	200	109218.05	109172.89 (0.07)	109059.48
MDG-b-31	2000	200	109061.67	109111.08 (0.07)	109028.62
MDG-b-32	2000	200	109121.63	109034.18 (0.06)	109237.88
MDG-b-33	2000	200	109131.86	109246.83 (0.08)	109116.12

Table 1.IV : Best Known Solutions for All the Datasets (Continued)

Instance	n	p	All methods	Multiple runs	Long run
MDG-b-34	2000	200	109055.53	109088.63 (0.06)	109044.15
MDG-b-35	2000	200	109198.53	109169.59 (0.06)	109078.56
MDG-b-36	2000	200	109146.42	109263.37 (0.08)	109211.59
MDG-b-37	2000	200	109190.1	109269.27 (0.1)	109166.56
MDG-b-38	2000	200	109201.61	109203.94 (0.07)	109061.18
MDG-b-39	2000	200	109133.48	109151.91 (0.07)	109161.22
MDG-b-40	2000	200	109069.51	109225.36 (0.06)	109273.17
Average			109108.8	**109170.14**	109113.47
MDG-c-1	3000	300	161227	161046 (0.05)	161227
MDG-c-2	3000	300	161065	160957 (0.06)	161014
MDG-c-3	3000	300	160868	160943 (0.05)	160949
MDG-c-4	3000	300	161092	161466 (0.14)	160982
MDG-c-5	3000	300	160883	160950 (0.08)	160883
MDG-c-6	3000	400	211407	211378 (0.05)	211530
MDG-c-7	3000	400	211344	211467 (0.05)	211612
MDG-c-8	3000	400	211391	211391 (0.06)	211475
MDG-c-9	3000	400	211308	211474 (0.06)	211278
MDG-c-10	3000	400	211359	211533 (0.07)	211365
MDG-c-11	3000	500	261292	261243 (0.05)	261486
MDG-c-12	3000	500	261242	261232 (0.06)	261412
MDG-c-13	3000	500	261393	261288 (0.03)	261481
MDG-c-14	3000	500	261563	261326 (0.04)	261450
MDG-c-15	3000	500	261599	261337 (0.04)	261748
MDG-c-16	3000	600	311381	311042 (0.1)	311678
MDG-c-17	3000	600	311283	311031 (0.12)	311256
MDG-c-18	3000	600	311264	310895 (0.09)	311152
MDG-c-19	3000	600	311310	310972 (0.16)	311613
MDG-c-20	3000	600	311384	311023 (0.12)	311350
Average			236282.75	236199.7	**236347.05**

The second question addresses longer running times for the heuristics in hope of finding higher quality solutions. Here we take the same approach and run the preliminary experiments only on the MDG-c largest dataset. The above-mentioned three heuristics are run only once but this time with 10 hours of running time. If we compare the average of the solutions over the 20 instances of the MDG-c dataset, the long run *(GD-SG-p)* heuristic

makes a 0.9% improvement compared to its short run version. This improvement is 0.5% and 0.3% for the *(GD-SG-0.75p)* and *(RA-SG-p)* methods respectively. Of course the calculation of the difference between the short run and long run of the greedy start heuristics makes more sense since they are more robust, and it is harder to do so for the random start versions as they could lead to a different solution right from the start regardless of their running time. Yet, for the rest of the three datasets the *(GD-SG-p)* heuristic is chosen for the long run experiments as it is more likely to lead to higher quality solutions. The results of the long run of the *(GD-SG-p)* are presented in the last column (*Long run*) of Table 1.IV for each dataset. Comparison of the fourth and the sixth columns of Table 1.IV reveals that the long run experiment has led to 4, 10 and 11 new improvements for the MDG-a, MDG-b and MDG-c datasets respectively. It should be noted that for some data instances in Table 1.IV , the best value obtained in the fourth column (*All methods*) is higher than the best solutions in the fifth and the sixth columns. The reason is that the values reported in the fourth column correspond to the best values under a *single* run of all the previously discussed 72 combinations for two hours of running time, whereas the fifth column reports the best values after 30 runs of only the *(RA-SG-p)* heuristic for two hours of running time, and the sixth column represents the values of a *single* run of only the *(GD-SG-p)* heuristic for 10 hours of running time.

1.3.3 Comparison With Exact Methods

This is the first application of any heuristic method on large p-dispersion-sum problem instances. In order to verify the quality of the solutions obtained by our heuristics we use the ILOG CPLEX 12.4 for exact solutions. Yet, even the smallest instances seem impossible for exact methods to solve in a reasonable time. As the result we provided ILOG CPLEX with the best solutions ever obtained from our heuristics in Section 1.3 as an initial solution and then calculated an upper bound in order to facilitate the problem resolution.

The upper bound is calculated as follows: for each of the n vertices their distances to all other vertices is sorted in decreasing order, then the sum of the distances to their furthest $(p-1)$ vertices is calculated. As the result for each vertex there is a value that represents the sum of the distances to its most distant $(p-1)$ vertices. Now if these values are sorted

in decreasing order, it is assured that the optimal solution can never exceed the value in the p_{th} rank in this sorted list.

After having provided the initial solution and the upper bound for ILOG CPLEX, we ran it only on the smallest SOM-b dataset and let it run as long as ILOG CPLEX is capable up to a maximum of two weeks running time. In the fourth column of Table 1.V the best solution ever obtained from all heuristics are given as lower bound, in the fifth column the above-mentioned upper bound is given for each instance, in the sixth column the best bound by ILOG CPLEX is presented and finally in the last column the best obtained Gap is given. As it is seen even after having provided a lower and upper bound and such a long running time, ILOG CPLEX is never capable of improving our solution. This confirms the quality of our solutions obtained and the complexity of such problems for exact methods.

Table 1.V : Comparison With Exact Methods

Instance	n	p	Lower bound	Upper bound	Best bound	Gap
SOM-b-1	100	10	62	81	62 (7sec)	optimal
SOM-b-2	100	20	111	164	111 (3hr)	optimal
SOM-b-3	100	30	151	234	151 (52.16hr)	optimal
SOM-b-4	100	40	195	293	195 (12.87hr)	optimal
SOM-b-5	200	20	117	171	122 (336hr)	4.27%
SOM-b-6	200	40	212	335	235 (336hr)	10.85%
SOM-b-7	200	60	298	475	331 (336hr)	11.07%
SOM-b-8	200	80	386	596	423 (336hr)	9.58%
SOM-b-9	300	30	170	261	195 (336hr)	14.7%
SOM-b-10	300	60	309	506	365 (336hr)	18.12%
SOM-b-11	300	90	440	716	516 (336hr)	17.27%
SOM-b-12	300	120	572	898	657 (336hr)	14.86%
SOM-b-13	400	40	222	351	263 (336hr)	18.46%
SOM-b-14	400	80	405	677	493 (336hr)	21.72%
SOM-b-15	400	120	580	958	698 (336hr)	20.34%
SOM-b-16	400	160	752	1195	882 (336hr)	17.28%
SOM-b-17	500	50	272	441	334 (336hr)	22.79%
SOM-b-18	500	100	503	847	625 (336hr)	24.25%
SOM-b-19	500	150	726	1199	892 (336hr)	22.87%
SOM-b-20	500	200	937	1497	1115 (336hr)	19.00%

1.4 Conclusions and Future Work

In this work we presented a general `VNS` framework for the p-dispersion-sum problem, which to the best of our knowledge is the first application of any heuristic for this variation of dispersion problems. We then presented a detailed experimental setting which captured a vast number of possibilities within the `VNS` framework. In the results and analysis sec-

tion we addressed the tradeoffs between the greedy intensification modules versus the more random diversification techniques embedded in VNS. We believe that the more intensified modules lead to overall higher quality solutions, whereas the more diversified modules increase the chances of obtaining new improvements only if several repetitions of the heuristics are allowed. The greedy approaches are more robust and their repeated runs do not seem a promising approach, on the contrary one can expect further improvements by allowing longer running times for such heuristics.

One of the most interesting advantages of the Variable Neighborhood Search metaheuristic is its flexibility and how it allows the decision maker to define and adapt the framework to its own problem specifications. The choice of the best setting is always a matter of time and available computational resources and also the fact that if one is interested in a heuristic that provides more robust and higher quality solutions on average, or a method that gives the opportunity of obtaining new improved solutions over repeated runs.

We were specifically interested in studying the behavior of main VNS components and tried to avoid tuning of its parameters. Of course it is plausible to expect new improvements in the best solutions obtained in this work by further tuning the maximum shake size parameter which could serve as an idea to be investigated in the future VNS-based or other suitable heuristic methods for dispersion problems.

Acknowledgements

This research was funded by NSERC (Natural Sciences and Engineering Research Council of Canada) grant PGSD2-392404-2010, and FQRNT (Fonds de recherche du Québec - Nature et technologies) grant 134582. Pierre Hansen has been partially supported by NSERC grant 105574-2007, Sylvain Perron has been partially supported by NSERC grant 327435-06, and they were both partially supported by FQRNT team grant PR-131365.

Chapter 2

A Greedy Variable Neighborhood Search Heuristic for the MaxSumSum p-Dispersion Problem

Behnaz Saboonchi

Pierre Hansen

Sylvain Perron

Department of Management Sciences

GERAD and HEC Montréal

3000, chemin de la Côte-Sainte-Catherine

Montréal, Québec, H3T 2A7, Canada

Abstract

The MaxSumSum (maximum diversity) problem consists of the selection of p facilities among n candidate locations in a way that the total sum of the distances between each pair of the located facilities is maximized. The Basic Variable Neighborhood Search heuristic (`BVNS`) has already been applied to solve this problem with success. In this work we have developed a Greedy Variable Neighborhood Search heuristic which adds a new type of plateau search mechanism to its general framework. This newly incorporated local search technique helps the exploration of the solution space and facilitates finding higher quality solutions. The proposed solution procedure further improves the already high performance of the `BVNS` and finds new improved solutions for several of the largest benchmark datasets in the literature.

2.1 Introduction

In the family of the dispersion problems, given a set of n vertices one intends to select a subset of size p in a way that a function of the distance among the selected vertices is maximized. This is useful when some measure of diversity in the solutions is desirable. For instance in the logistics context it can be used in the location of missile silos where dispersion can reduce the chances of all of them being attacked or for locating obnoxious facilities to be far from population zones [32]. The dispersion can also be a desirable factor when it comes to franchise location problems where managers intend to avoid the cannibalization effects within the chain. The difference is not always translated into the physical distance. For instance dispersion models can also be used in order to design a portfolio of new products where it is desirable to enter the market with a group of products which are as dissimilar as possible in terms of the quality, price, shape, etc. Another example would be in multi-objective problems where the decision maker may be interested in selecting a collection of solutions as far as possible for each objective [72].

Erkut and Neuman [33] propose four different types of the dispersion models based on different dispersion metrics. The first one is the MaxMinMin problem which maximizes the minimum distance between each pair of facilities. The second one is the MaxSumMin which seeks to maximize the sum of the minimum distances from each facility to its closest neighbor. The third formulation is called MaxMinSum which takes the sum of the distances

from each facility to all its neighbors and maximizes the smallest sum of the distances. Finally the fourth formulation corresponds to the MaxSumSum which aims at maximizing the sum of all the hub distances for all located facilities. This model tries to locate p facilities far from a given set of nodes and far from each other and is the one studied in this paper.

Prokopyev et al. [71] introduce three equity-based measures for the dispersion problems and propose a solution procedure based on the Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic. Its main components are the construction phase where a complete solution is created by a greedy addition of the components of a solution, and an improvement phase which performs local perturbations to get a local optimal solution.

Hansen and Moon prove that the discrete version of the maximum diversity (MaxSumSum p-dispersion) problem on general networks is strongly NP-complete, by reduction to the stable set problem [11, 33, 59, 63, 80]. Yet, metaheuristics have shown to be very successful in finding high quality solutions to this problem and have been widely discussed and compared in the literature [4, 59, 63]. An extensive comparison is done by Martí et al. [63] where they compare 10 heuristics and 20 applications of metaheuristics for this problem. They conclude that the Basic Variable Neighborhood Search (`BVNS`) method by Brimberg et al. [11] and the Iterated Tabu Search (ITS) method by Palubeckis [67] are the most powerful applications of metaheuristics for this problem.

The ITS [67] applies a tabu search step followed by a local search in case a better incumbent solution has been found. Then a perturbation procedure is applied by swapping a random number of selected and unselected points which is similar to the shake procedure in `VNS` except that the shake size is random at each iteration.

The `VNS` method applied by Brimberg et al. [11] to the heaviest k-subgraph problem (HSP) can also be adapted to solve the MaxSumSum problem. In fact HSP is more general than the MaxSumSum problem as the edge weights do not need to represent distances and the graph is not necessarily fully connected. They compare the basic `VNS` (`BVNS`), skewed `VNS` (`SVNS`) and the greedy add and greedy drop construction procedures followed by `VNS`, and finally conclude that the `BVNS` is the overall best method. The `BVNS` consists of a random shaking perturbation strategy followed by a local search construction phase. Their data structure allows an efficient update of the values and thus has also been applied in

this paper. The BVNS is among the most successful methods to address the MaxSumSum problem and in this work we further improve the capabilities of the VNS by developing more elaborate modules.

Another recent successful method is the Iterated Greedy metaheuristic (IG) and its fine-tuned version (TIG) by Lozano et al. [59] which generates a sequence of solutions by iterating over greedy construction and destruction phases. This method has not been considered in the comprehensive literature review of Martí et al. [63].

Finally, Wang et al. [80] compare the most successful candidate methods for the MaxSum-Sum problem with their Learnable Tabu Search method guided by Estimation of Distribution Algorithm (LTS-EDA). Their method can extract knowledge during the tabu search procedure and adapts the search structure. The clustered EDA is a learnable constructive method in order to create new starting solutions, coupled with the TS as an improvement method. They use either some of the executable codes provided by various authors or code the suggested algorithms in the literature themselves, and compare all the metaheuristics including the ITS, VNS, TIG and LTS-EDA under the same conditions. Their final comparison shows that the LTS-EDA obtains the best improvements over the existing large benchmark test problems. The best results are obtained in the long run version of the experiments, that will be considered as the best known solutions in the literature in Section 2.4.

In Section 2.2 we describe the problem in graph theoretical terms followed by its mixed integer formulation. Section 2.3 presents a detailed explanation of our proposed greedy VNS heuristic solution procedure for the MaxSumSum p-dispersion problem. Then we discuss our computational experiments on the largest known benchmark test problems and finally conclude the paper by highlighting our contributions and suggestions for future research.

2.2 Problem Statement and Mathematical Formulation

This section expresses the MaxSumSum p-dispersion problem in graph theoretical terms and then presents its respective mathematical formulation. Let $V = \{v_i, \forall i = 1, \ldots, n\}$, be a set of n vertices (potential locations) and v_i representing each member of this set. Let E be the set of $\binom{n}{2}$ edges of an undirected and fully connected graph $G(V, E)$, with $d_e > 0$ representing the distance over each edge $e \in E$. The value p is an integer such that

$3 \leq p \leq |V|$. We define S as any subset of p vertices such that $S \subseteq V, |S| = p$. The subset of the vertices not present in the current solution is defined as $\bar{S}$ such that $\bar{S} = V \setminus S, |\bar{S}| = n-p$.

The objective function value $f(S)$ is defined as the total sum of the distances among all the p selected vertices induced by the subset S:

$$f(S) = \sum_{e \in E(S)} d_e.$$

The MaxSumSum problem intends to find the optimal subgraph $G(S^*, E(S^*))$, where:

$$S^* = \arg\max_S f(S).$$

This problem could be modeled as the following 0-1 mixed integer program as suggested in [33]:

$$\begin{aligned}
\max \quad & \sum_{i=1}^{n} Z_i & \\
s.t. \quad & Z_i \leq M x_i & 1 \leq i \leq n \\
& Z_i \leq \sum_{j=1}^{n} d(v_i, v_j) x_j & 1 \leq i \leq n \\
& \sum_{i=1}^{n} x_i = p & \\
& x_i \in \{0, 1\} & 1 \leq i \leq n,
\end{aligned}$$

where x_i is a binary decision variable defining if vertex v_i is selected, $d(v_i, v_j)$ is the distance between any pair of the located facilities at locations i and j and M is a sufficiently large number which could be set as the sum of the p largest distances among the n vertices. The distances between all the vertices are taken as an input and stored in an $n \times n$ upper-triangular matrix with $d(v_i, v_i) = 0$.

2.3 VNS for the *p*-Dispersion-Sum Problem

Variable Neighborhood Search (`VNS`) is a metaheuristic or framework for building heuristics which is based on the idea of a systematic change of the neighborhood in order to escape

from the valleys surrounding local optima, followed by a local search to find improved solutions. This general method has been proposed by Mladenović and Hansen [64] and has proven to lead to very successful heuristics for solving large combinatorial programs with applications in location theory, cluster analysis and several other fields. For a recent survey of the theoretical developments and applications including several hundred references see [45, 48].

The Basic Variable Neighborhood Search method (`BVNS`) has already been applied to the heaviest k-subgraph problem and has shown to be among the most efficient methods compared to other heuristics for the maximum diversity problem [11, 63]. Palubeckis et al. [68] also apply the multistart simulated annealing, hybrid genetic and variable neighborhood search algorithms to solve the MaxSumSum problem. Their computational experiments on benchmark instances of size up to 2000 elements show that there is no clear winner in all cases, however, the comparison of the heuristics on larger instances favors the VNS algorithm.

Brimberg et al. [11] suggest the examination of different `VNS` strategies and also extensive experimental testings of `VNS` on different types of graphs as a future extension to their work. Therefore, we have decided to develop a more elaborate heuristic method within the VNS framework that is well-suited to the MaxSumSum p-dispersion problem.

In order to represent the solution at each step of the heuristic we use the data structure suggested in [11]. The solution is represented by an array of the n indices corresponding to each vertex or candidate location, where the first p elements correspond to the set of the current solution S.

The solution space U is represented by the $\binom{n}{p}$ subsets of V with cardinality p. In order to apply `VNS` a metric function is defined to evaluate the distance between any two solutions S and S':

$$\delta(S, S') = \delta(S', S) = |S \setminus S'|.$$

Based on the metric distance function defined above, the neighborhood of size k of a solution S is defined as:

$$N_k(S) = \{S' \in U | \delta(S, S') = k\}; k = 1, 2, \ldots, \min\{p, n - p\}.$$

Throughout this paper the following notations are used:

- $S_{best/cur}$: the best/current solution set corresponding to the best/current objective function value;
- $f(S_{best/cur})$: the best/current objective function value corresponding to the sum of the distances among all the selected vertices in the best/current solution set $S_{best/cur}$;
- $W(v_i)$: the sum of the distances from any vertex v_i $(i = 1, \ldots, n)$ to all the vertices in the solution set S;
- v_{exit}: the vertex inside the solution set that is a candidate to *leave* the solution set $(v_{exit} \in S)$;
- v_{enter}: the vertex outside the solution set that is a candidate to *enter* the solution set $(v_{enter} \in \bar{S})$.

These values are first computed at the construction of the initial solution and are updated each time a new solution is found.

In Algorithm 2.1 we define our `VNS` function and then in the following sections we explain in details the functions embedded in our general framework. The stopping criterion is the total execution time t_{max} and the already elapsed cumulative time in the overall procedure is denoted by $t_{elapsed}$. The k_{min} and k_{step} (step size) parameters are set by default to 1, and the k_{max} (maximum shake size) is set to a coefficient of $\min\{p, n-p\}$, as discussed later.

2.3.1 Initialization

The initial solution could be created at random or in a greedy manner. Based on the *random* method the initial solution is simply created by choosing p vertices at random.

Two *Greedy* concepts have been widely used in the literature in order to create initial solutions for the dispersion problems [35]. The *Greedy deletion* heuristic starts with all the n vertices and eliminates one vertex at each iteration. For this problem the deletion candidate is the one with the smallest sum of the distances value with the rest of the remaining vertices at each iteration, and the ties are broken arbitrarily. Of course this procedure is repeated $(n-p)$ times until exactly p vertices remain in the solution set.

```
function VNS (k_min, k_step, k_max)
S_cur ← Initialize();
S_best ← S_cur;
t_elapsed = 0;
k_max = min{p, n − p};
while t_elapsed ≤ t_max do
    k_cur ← k_min;
    while k_cur ≤ k_max and t_elapsed ≤ t_max do
        S_cur ← Shake(S_cur);
        S_cur ← LocalSearch(S_cur);
        S_cur ← RefinedLocalSearch(S_cur);
        if f(S_cur) > f(S_best) then
            S_best ← S_cur;
            k_cur ← k_min;
        else
            k_cur ← k_cur + k_step;
        end
    end
end
```

Figure 2.1: Pseudo Code for the VNS Framework

The *Greedy add* heuristic selects a starting vertex at random and creates the complete solution set in $(p-1)$ iterations. As the result, the size of the under construction solution set is smaller than p and will gradually reach the complete size as the construction phase is executed. If we represent the solution set under construction (i.e., the size of the set is smaller than p) as C, and the set of the vertices outside this set as $\bar{C}$, the entering candidate $v_i \in \bar{C}$ would be the one with the largest $W(v_i)$ value (the sum of distances to the vertices in the under construction solution C) [4, 59]. At each iteration the objective function value is the total sum of the $W(v_i)$ values for all the vertices $v_i \in C$. After the addition of the entering vertex $v_{enter} \in \bar{C}$, the existing sum of the distances values for the under construction solution vertices will be updated as: $W(v_i) + d(v_i, v_{enter})$ for all the $v_i \in C$. Therefore, the objective function value after the addition of each v_{enter} will be: $f(C) + \sum_{v_i \in C} d(v_i, v_{enter})$.

This heuristic could be repeated n times based on different starting vertices and then the best one leading to the highest objective value could be selected. Based on preliminary results we know that this heuristic is very time consuming (much more than the *Greedy deletion* heuristic), and for large datasets it is not worthwhile to take this procedure just to further improve the initial solution. In order to overcome this drawback the *Greedy*

add heuristic is initialized by choosing the two furthest vertices as the initial vertices and by repeating the above-mentioned procedure $(p-2)$ times. We observed empirically that the results obtained by this method are among the highest possibilities for the *Greedy add* heuristic without spending too much computational time on the initial solution.

2.3.2 Local Search

After having created the initial solution, the `LocalSearch` procedure is called performing 1-interchange swaps on the current solution as shown in Algorithm 2.2. This means that at each iteration only *one* pair of vertices is swapped at a time. The swap could be done whenever the first (first improvement strategy) or the best (best improvement strategy) contribution is made to the current objective value. In order to avoid unterminated loops, only positive contributions would induce a swap.

In order to execute the `LocalSearch` procedure the gain obtained from swapping the selected entering candidate with the selected leaving candidate should be evaluated. The two main `Contribution` and `Update` functions will be explained in details in the following subsection.

```
function LocalSearch(S)
repeat
    (v_exit, v_enter, gain) ← Contribution(S);
    if gain > 0 then
        Swap(v_exit, v_enter);
        Update(v_exit, v_enter, S);
    end
until gain > 0;
```

Figure 2.2: Pseudo Code for the Local Search Procedure

2.3.2.1 Contribution and Update

In the proposed `LocalSearch` procedure the `Contribution` function can determine the first or the best swapping pair ($v_{exit} \in S$ and $v_{enter} \in \bar{S}$), making a positive contribution to the current objective function value.

In order to initiate the `Contribution` function a random leaving candidate $v_{exit} \in S$, and a random entering candidate $v_{enter} \in \bar{S}$ are chosen and then the algorithm will move on to the subsequent vertices until a positive contribution is found. The change in the objective function value resulted from the swap will be calculated as :

$$change \leftarrow W(v_{enter}) - W(v_{exit}) - d(v_{exit}, v_{enter}).$$

With the first improvement strategy the `Contribution` function stops as soon as an improving solution is found, as the result in the worst case it is implemented in $O(p(n-p)) = O(np)$ time per `LocalSearch` iteration.

The `Update` procedure performs all the required updates before proceeding to the subsequent iteration. It is implemented in two different phases in order to update all the $W(v_i)$ values, and then to update the objective function value $f(S_{cur})$. The update for the $W(v_i)$ $(\forall i = 1, \ldots, n)$ is straightforward and is performed in $O(n)$ total time, whereas the update of the objective function is done in $O(1)$ time at each iteration. Thus the updated values would be:

$$W(v_i) = W(v_i) + d(v_i, v_{enter}) - d(v_i, v_{exit}), i = 1, 2, \ldots, n,$$

$$f(S_{cur}) = f(S_{cur}) + change.$$

2.3.3 Refined Local Search

The existence of plateaus or the flat landscapes of the MaxSumSum problem make it more difficult for the ascent procedures to find an improvement in the objective function especially as the size and the density of the graphs increase [11]. This has inspired us to develop a plateau search mechanism within the `VNS` metaheuristic framework.

The `RefinedLocalSearch` procedure demonstrated in Algorithm 2.3 is efficient both in the presence and the absence of plateaus. The reason is that it makes all the possible swaps that will improve the quality of the solution set by exchanging the low quality vertices with the ones of higher quality which will facilitate further improvements by the `LocalSearch`

procedure in the subsequent iterations. It should be noted that in Algorithm 2.3 the smallest $W(v_{exit}) \in S$ is found in $O(p)$ time and the largest $W(v_{enter}) \in \bar{S}$ is determined in $O(n-p)$ time.

Of course the `RefinedLocalSearch` procedure never worsens the actual solution and could even lead to improvements in the objective function value if the whole plateau is removed. As demonstrated later in Section 2.4, the addition of this module improves significantly the quality of the obtained solutions.

function `RefinedLocalSearch(`S`)`
$change = 0$;
while $change \geq 0$ **do**
    Find $v_{exit} \in S$ with the smallest $W(v_{exit})$;
    Find $v_{enter} \in \bar{S}$ with the largest $W(v_{enter})$;
    $change \leftarrow W(v_{enter}) - W(v_{exit}) - d(v_{exit}, v_{enter})$;
    if $change \geq 0$ **then**
        `Swap(`v_{exit}, v_{enter}`)`;
        `Update(`v_{exit}, v_{enter}, S`)`;
    end
end

Figure 2.3: Pseudo Code for the Refined Local Search Procedure

2.3.4 Shake

The perturbation in most `VNS`-based heuristics is done in a simple manner by choosing a random pair of vertices from S and $\bar{S}$ and then repeating k times the random swap move. The `RandomShake` function does so by choosing one random leaving and entering candidate at each iteration with updates in between each swap. However, here we have developed two additional shake functions in order to control the perturbation operation in a more intelligent manner.

The `SemiGreedyShake` function fixes a random leaving candidate from the current solution set S and then chooses an entering candidate that has the highest sum of the distances value in case it replaces the exit candidate, i.e., the highest $W(v_{enter}) - d(v_{exit}, v_{enter})$ value. As the result this shake method does not guarantee that a deterioration in the objective function value would not occur, yet it simply chooses a reasonable entering candidate after

having fixed the leaving candidate. This procedure is repeated until the shake size of k is attained. Each iteration is performed in $O(n-p)$ time and after each swap the `Update` function is called.

In order to have a more intensified shake operation the `GreedyShake` function has been developed which for a shake of size k, selects the k vertices with the smallest $W(v_i)$ values for all $v_i \in S$, and swaps all of them with the k vertices with the largest $W(v_i)$ values for all $v_i \in \bar{S}$. This greedy fashion of selecting the entering candidates could provide better starting solutions for the subsequent `LocalSearch` procedure. The k leaving and entering candidates are chosen all at once and are not changed while the updates are performed between the swaps. The performance of the three shake functions will be compared in details in Section 2.4 where it is demonstrated that the more intelligent shake functions dominate the purely random ones.

2.4 Computational Experiments

In this section we have selected four sets of the largest benchmark instances that were used for comparison purposes in the maximum diversity problem literature leading to 50 instances in total [11, 59, 63, 67, 80]. A brief description of the characteristics of the datasets is given below:

- MDG-a: this dataset consists of 20 matrices with real numbers randomly selected between 0 and 10 from a uniform distribution by Duarte and Martí [27] with $n = 2000$ and $p = 200$.
- MDG-c: this dataset consists of 20 matrices with $n = 3000$ and $p = 300$ (MDG-c-1 to 5), 400 (MDG-c-6 to 10), 500 (MDG-c-11 to 15) and 600 (MDG-c-16 to 20) [27].
- p5000 and p3000: these datasets consist of 10 matrices each with integer numbers generated from 0 to 100 from a uniform distribution [67]. There are five instances with $n = 3000$ and $p = 0.5n$, and five instances with n =5000 and $p = 0.5n$ with a matrix density of 10, 30, 50, 80 and 100%.

First we describe our experiments that were designed to study the performance of different settings within the `VNS` framework and then compare and analyze the tradeoffs and

overall results obtained by different methods over all the test problems. Finally we tune our framework based on the preliminary results and compare the results with those of the state-of-the-art heuristics in the literature.

All the heuristics were coded in C ++ and run on a linux machine with an intel processor of 2.667 GHz and 3Gb Ram. The best results obtained after two hours of running time are reported as suggested in [63].

2.4.1 Preliminary Experiments Setup

As mentioned in Section 2.3 our `VNS` implementation allows various settings and methods within its framework. In order to initialize the `VNS` three different methods have been discussed: Random add (RA), Greedy add (GA) and Greedy deletion (GD). There are also three different shaking possibilities: Random shake (RS), Semi-Greedy shake (SG) and Greedy shake (GS). In the general framework presented in Section 2.3 the shake size at each iteration increases systematically and will be reset to k_{min} whenever an improvement is made or when the k_{max} value is reached. The k_{max} is a parameter whose value by default is $\min\{p, n-p\}$, which could be a large value depending on the problem size. As the result two smaller values, i.e., $0.5 * \min\{p, n-p\}$ and $0.75 * \min\{p, n-p\}$ are tested as well to verify if it helps improve the performance of the heuristic. Besides each experiment could be done with or without the `RefinedLocalSearch` module. This will lead to $3 \times 3 \times 3 \times 2 = 54$ different `VNS` configurations.

On the other hand at each iteration of `VNS` either an improvement is made or not. In case of no improvement a decision on how to start the next iteration should be made. The next iteration is either started form the already best solution obtained (from best or FB), or from the current solution just obtained (from current or FC). The former will lead to more intensification in the search, whereas the latter favors diversification. Besides, the `LocalSearch` procedure can pursue a first improvement strategy ($FirstI$) favoring more diversification, versus best improvement strategy ($BestI$) leading to more intensification. The above mentioned intensification and diversification strategies will lead to four general `VNS` frameworks. As the result the datasets are used in the $54 \times 4 = 216$ total configurations.

We do not allow longer running times in order to get further improvements, as the result the tests are executed *only once* under *two hours* of running time [63]. Throughout the paper we present the average % *deviation* from the best solutions obtained by the 216 combinations for each group of dataset:

$$\% \text{ deviation} = \frac{\text{best obtained value - actual value}}{\text{best obtained value}} \times 100.$$

Table 2.I : Average % Deviation for Individual Methods

	MDG-a	MDG-c	p3000	p5000	Average
General framework					
FC-FirstI	**0.72**	**0.45**	0.24	**0.19**	**0.4**
FC-BestI	0.99	0.61	0.38	0.31	0.57
FB-FirstI	0.81	0.5	**0.22**	**0.19**	0.43
FB-BestI	1	0.64	0.32	0.27	0.56
Initialization					
RA	5.57	0.96	0.66	0.56	1.94
GA	2.85	0.42	0.14	0.11	0.88
GD	**2**	**0.29**	**0.07**	**0.05**	**0.6**
Shake method					
RS	6.34	1.07	0.74	0.62	2.19
SG	2.49	0.35	0.07	0.06	0.74
GS	**1.73**	**0.24**	**0.06**	**0.04**	**0.52**
Shake max size					
0.5p	**3.41**	**0.54**	**0.28**	**0.24**	**1.12**
0.75p	3.51	0.56	0.29	**0.24**	1.15
p	3.63	0.57	0.3	**0.24**	1.19
Refined local search					
Yes	**1.25**	**0.19**	**0.03**	**0.03**	**0.38**
No	5.79	0.92	0.55	0.45	1.93

Table 2.I represents the average deviation from the best solutions obtained for all the 50 data instances for each of the *four* `VNS` general frameworks, *three* initialization methods, *three* shaking strategies, *three* possible k_{max} sizes, and all with either the presence or the absence of the `RefinedLocalSearch` module. The average deviations are all calculated based on the results presented under the G-VNS column of Tables 2.II to 2.IV which refer to the best solutions obtained by a *single* run of the 216 combinations. The smallest average deviation values for each group are shown in bold under the *Average* column of Table 2.I . Same presentation has also been done to highlight the smallest average deviation values for each of the four datasets.

The first part of Table 2.I which refers to the four general `VNS` frameworks reveals that the current iteration strategy and first contribution local search strategy *(FC-FirstI)* leads

to the overall lowest average deviation. This is also true for each individual dataset except for the p3000 instances where the *(FB-FirstI)* strategy leads to the lowest average deviation. In the second part which refers to different initial solution generation methods, the Greedy deletion strategy (*GD*) consistently leads to the lowest average deviation for all datasets. In the third group which addresses the shaking strategies, the Greedy shake (*GS*) strategy has the lowest average deviation value across all the datasets which emphasizes the importance of more intelligent shaking strategies compared to pure random ones. The only parameter that we changed in our experiments is the maximum shake size which is presented in the last comparison group. As it is seen the smaller maximum shake size is slightly better for all instances, yet the difference among the results (average deviation value) is not significant. Finally in the last part it is clearly observed that the existence of the `RefinedLocalSearch` module leads to significantly lower average deviations which highlights the importance of the addition of this new local search mechanism to the classical body of `VNS`.

To summarize, four important observations can be highlighted:

- The first improvement Local Search strategies (*FC-FirstI* and *FB-FirstI*) lead to the lowest average deviations compared to other general frameworks.
- The Greedy deletion (*GD*) initialization method leads to the highest average quality.
- The Greedy Shake (*GS*) method leads to the lowest average deviation over all data instances.
- The inclusion of the `RefinedLocalSearch` plateau search mechanism contributes significantly to the creation of high quality solutions.

2.4.2 Results and Analysis

The best preliminary results obtained in Section 2.4.1 are presented under the Greedy `VNS` (G-VNS) column in Tables 2.II to 2.IV . Based on the findings in the previous section, the following `VNS` setting is selected in order to do the final tuned experiments, called the Tuned-Greedy `VNS` (TG-VNS):

- Each iteration will start form the current solution just obtained (the current iteration strategy), and the Local Search function will randomly choose between the first or best improvement strategies.
- The Random add (RA) initialization method is selected in order to obtain more diversified solutions in repeated runs. The Greedy deletion (GD) strategy leads to robust high quality solutions, yet based on our preliminary experiments obtains the same results in multiple runs.
- The Greedy shake (GS) method is used with the maximum shake size randomly selected as $0.5 * \min\{p, n-p\}$ or $0.75 * \min\{p, n-p\}$.
- The `RefinedLocalSearch` plateau search mechanism will be applied in all the experiments.

Table 2.II : Comparison of the Best Known Results for the MDG-a Instances

Instance	n	p	Best known	G-VNS	TG-VNS		
					Best	**Average**	**CV**
MDG-a-21	2000	200	114271 (ITS)	**0**	**0**	-3.33	0.00003
MDG-a-22	2000	200	114327 (ITS)	**0**	**0**	**0**	**0**
MDG-a-23	2000	200	114195 (ITS)	**0**	**0**	**0**	**0**
MDG-a-24	2000	200	114093 (ITS)	**0**	**0**	-5.2	0.00002
MDG-a-25	2000	200	114196 (ITS)	**0**	**0**	**0**	**0**
MDG-a-26	2000	200	114265 (ITS)	**0**	**0**	**0**	**0**
MDG-a-27	2000	200	114361 (ITS)	**0**	**0**	**0**	**0**
MDG-a-28	2000	200	114327 (ITS)	**0**	**0**	**0**	**0**
MDG-a-29	2000	200	114199 (ITS)	**0**	**0**	-1.87	0.000005
MDG-a-30	2000	200	114229 (ITS)	**0**	**0**	**0**	**0**
MDG-a-31	2000	200	114214 (ITS)	**0**	**0**	-6.6	0.00009
MDG-a-32	2000	200	114214 (ITS)	**0**	**0**	-7	0.00007
MDG-a-33	2000	200	114233 (ITS)	**0**	**0**	-3.73	0.00002
MDG-a-34	2000	200	114216 (ITS)	**0**	**0**	**0**	**0**
MDG-a-35	2000	200	114240 (ITS)	**0**	**0**	-0.87	0.000003
MDG-a-36	2000	200	114335 (ITS)	**0**	**0**	**0**	**0**
MDG-a-37	2000	200	114255 (ITS)	**0**	**0**	**0**	**0**
MDG-a-38	2000	200	114408 (ITS)	**0**	**0**	-1.73	0.00001
MDG-a-39	2000	200	114201 (ITS)	**0**	**0**	**0**	**0**
MDG-a-40	2000	200	114349 (ITS)	**0**	**0**	**0**	**0**

All the instances are run 15 times with the two hours of running time constraint under the above selected settings and the obtained solutions for all the 50 data instances are presented under the Tuned-Greedy `VNS` (TG-VNS) columns of Tables 2.II to 2.IV .

In the *Best Known* column of Tables 2.II to 2.IV the best known solutions for each of the instances are presented which have been found by different powerful heuristic methods

from various references. It should be noted that the best results in the literature have been obtained by longer running times of 5h for smaller instances and 10h for larger instances. Despite the fact that our running time is only two hours, we still keep these best solutions as a benchmark in order to verify the quality of our solutions.

Table 2.III : Comparison of the Best Known Results for the MDG-c Instances

Instance	n	p	Best known	G-VNS	TG-VNS		
					Best	Average	CV
MDG-c-1	3000	300	24924685 (BVNS)	**0**	**+1659**	-270.33	0.003
MDG-c-2	3000	300	24909199 (BVNS)	**+3347**	**+3347**	**+2069.13**	0.005
MDG-c-3	3000	300	24900820 (ITS)	**+4398**	**+4398**	**+2266.93**	0.011
MDG-c-4	3000	300	24904964 (BVNS)	**+4746**	**+4746**	**+916.4**	0.006
MDG-c-5	3000	300	24899703 (ITS)	-3999	-3999	-5139.4	0.005
MDG-c-6	3000	400	43465087 (ITS)	-20139	-20139	-24533.47	0.008
MDG-c-7	3000	400	43477267 (BVNS)	**0**	**0**	-737.13	0.002
MDG-c-8	3000	400	43458007 (BVNS)	**+7565**	**+7565**	**+2944.33**	0.005
MDG-c-9	3000	400	43448137 (BVNS)	**0**	**0**	-394.87	0.001
MDG-c-10	3000	400	43476251 (ITS)	-10690	-10690	-10782.73	0.001
MDG-c-11	3000	500	67009114 (BVNS)	**+12018**	**+12018**	**+6720.4**	0.006
MDG-c-12	3000	500	67021888 (ITS)	-7718	-7718	-9397.87	0.006
MDG-c-13	3000	500	67024373 (BVNS)	**0**	**0**	-392	0.001
MDG-c-14	3000	500	67024804 (BVNS)	**+5386**	**+5386**	**+3395.87**	0.004
MDG-c-15	3000	500	67056334 (BVNS)	-1624	**0**	-2448.2	0.004
MDG-c-16	3000	600	95637733 (BVNS)	**+1196**	**+1196**	-174.53	0.001
MDG-c-17	3000	600	95645826 (ITS)	-74713	-74713	-76052	0.002
MDG-c-18	3000	600	95629207 (ITS)	-97066	-97100	-100356.2	0.005
MDG-c-19	3000	600	95633549 (ITS)	-34385	-34385	-35069.8	0.001
MDG-c-20	3000	600	95643586 (ITS)	-59104	-59104	-59509.07	0.001

The *G-VNS* column represents the proposed Greedy `VNS` results which show the difference between the best solutions obtained in the preliminary experiments of 216 `VNS` module combinations presented in Section 2.4.1 and the best known solutions in the literature. A *positive* value represents an improvement over the best known solutions reported, a value of *zero* means that we have equaled the best ever value and a *negative* value means that our solution is inferior to the one reported in the literature. Here no average results are presented as in the preliminary experiments each combination is run only once.

The following three columns (Best, Average, CV) under the *TG-VNS*, represent the difference between the *Best* and *Average* results obtained by multiple runs of the Tuned Greedy `VNS` with the best known results in the literature. The reported *Best* values are the best among 15 runs of the *TG-VNS*, yet in order to have a better idea of the overall quality of all the 15 runs, the coefficient of variation *(CV)* is presented for each dataset under the *CV* column:

$$\text{CV} = \frac{\text{standard deviation}}{\text{mean}} \times 100.$$

As seen for all the 50 instances in Tables 2.II to 2.IV , the coefficient of variation values are generally very small, much inferior than 0.01%, which shows that our Random add (RA) methods are also very robust.

As illustrated by the bold values in Table 2.II , our proposed method finds *all* of the best ever solutions for the MDG-a instances by the G-VNS preliminary experiments and the TG-VNS *Best* case, and also 12 out of 20 in the TG-VNS *Average* case. The best solutions have been initially obtained by the Iterated Tabu Search (ITS) method in [67] and have also been found in the long run experiments of Tuned Iterated Greedy method (TIG) in [59] and the Learnable Tabu Search method (LTS-EDA) in [80]. Yet, none of these methods finds *all* the best solutions at once even in longer running times.

Most of the best results for the MDG-c instances in the literature have been obtained by the `BVNS` [11] and the ITS [67]. Our method finds equal or improved solutions for 11 instances by the G-VNS preliminary experiments, 12 instances by the TG-VNS *Best* case and 6 instances by TG-VNS *Average* case which are represented in Table 2.III .

Table 2.IV : Comparison of the Best Known Results for the p3000 and p5000 Instances

Instance	n	p	Best known	G-VNS	TG-VNS		
					Best	**Average**	**CV**
p3000-1	3000	1500	6502308 (LTS)	-208	**+22**	-422.13	0.01
p3000-2	3000	1500	18272568 (LTS)	-354	**0**	-362.93	0.001
p3000-3	3000	1500	29867138 (ITS)	**0**	**0**	-634.8	0.002
p3000-4	3000	1500	46915044 (LTS)	-96	-86	-429.33	0.001
p3000-5	3000	1500	58095426 (LTS)	-237	**+41**	-1101.4	0.002
p5000-1	5000	2500	17509215 (LTS)	-966	**+112**	-419.93	0.005
p5000-2	5000	2500	50102729 (LTS)	-1196	**+156**	-836.6	0.004
p5000-3	5000	2500	82039686 (LTS)	-1482	-426	-2746.73	0.003
p5000-4	5000	2500	129413112 (LTS)	-59	**+112**	-1878.13	0.001
p5000-5	5000	2500	160597781 (LTS)	-430	**+15**	-843.67	0.001

The best results in the literature for the p3000 and p5000 instances represent the best ever results obtained by state-of-the-art heuristics such as ITS, VNS, TIG and LTS-EDA under 5h and 10h running times. As shown in Table 2.IV , the G-VNS method finds one equal solution, and the TG-VNS *Best* case finds 8 out of 10 equal or better solutions, representing the best solutions in 15 runs of two hours of computation each.

2.5 Conclusions and Future Work

In this work we applied an elaborate greedy VNS framework over 50 of the largest data instances for the MaxSumSum p-dispersion problem. We first explained a detailed preliminary experimental setting which captured a vast number of possibilities within the VNS framework and then selected the most promising setting in order to conduct the tuned experiments. We also incorporated a new local search module within the VNS framework coupled with elaborate shake functions. Our extensive computational experiments on the largest benchmarks in the literature found new best solutions for several instances that proves the high performance of our method.

Of course in order to have a more precise comparison of various heuristics, the same executable codes of the different authors should be run under the same conditions. Yet, we believe that the reported results represent a fair comparison with other state-of-the-art heuristics, as we have considered the best ever solutions obtained by all the heuristic methods ever studied in the literature under long running times as our benchmark.

One of the most interesting advantages of the Variable Neighborhood Search metaheuristic is its flexibility and how it allows the decision maker to define and adapt the framework to its own problem specifications. The choice of the best setting is always a matter of time and available computational resources, and also the fact that if one is interested in a heuristic that provides more robust and higher quality solutions on average, or a method that gives the opportunity of obtaining new improved solutions over repeated runs. As the future work we suggest developing decomposed VNS frameworks [46] in order to tackle large problem instances in a more reasonable time and to possibly further improve the obtained results.

Acknowledgements

This research was funded by NSERC (Natural Sciences and Engineering Research Council of Canada) grant PGSD2-392404-2010, and FQRNT (Fonds de recherche du Québec - Nature et technologies) grant 134582. Pierre Hansen has been partially supported by NSERC grant 105574-2007, Sylvain Perron has been partially supported by NSERC grant 327435-06, and they were both partially supported by FQRNT team grant PR-131365.

Chapter 3

Franchise Location Models and Cannibalization Effects: A Variable Neighborhood Search Approach

Behnaz Saboonchi

Pierre Hansen

Sylvain Perron

Department of Management Sciences

GERAD and HEC Montréal

3000, chemin de la Côte-Sainte-Catherine

Montréal, Québec, H3T 2A7, Canada

Abstract

Application of the dispersion models in order to address the cannibalization phenomenon within franchised chains is a new approach. In this work we have developed a Variable Neighborhood Search (VNS) method in order to solve the MaxMinMin p-dispersion problem, which adds a new type of plateau search mechanism to the classical VNS metaheuristic framework. Besides, several other contributions have been made to the basic VNS heuristic in terms of the ascent and perturbation procedures. To the best of our knowledge this is the first application of the VNS to the MaxMinMin problem and our approach, compared to previous methods, finds or improves a results for *all* of the large-sized benchmarks within a shorter time and low computational efforts. Finding most of the proven optimal solutions in a fraction of a second, the robustness and quality of the solutions and the low complexity of the methods demonstrate the strength of the proposed heuristic solution procedures vis-à-vis the state-of-the-art algorithms for the MaxMinMin problem.

3.1 Introduction

Managing and locating retail and service networks have been widely addressed in different empirical and mathematical studies. The location decision process becomes more complex when it comes to franchise chains where the objective is not locating a single independent store, but a group of stores over time within a network. This location decision should be made without cannibalizing the existing same-brand units revenues. Encroachment and revenue cannibalization which are one of the main concerns of today's franchised chains occur when the franchiser adds the same-brand units in the proximity of the existing units.

The compromise between minimization of the chain cannibalization and the revenue maximization has been addressed in the literature. Ghosh and Craig [39] develop a location allocation model which evaluates the net impact of the new unit based on its cannibalization effects and the positive impacts it has on the whole system. They conclude that the conflicts will increase by more distance-sensitive customers or if we intend to locate multiple new facilities.

This issue could also be modeled as the minimization of cannibalization as a function of the difference of the market share of the existing units before and after the addition of

the new outlet. Fernandez et al. [36] take this approach in a multi-objective model which minimizes cannibalization and maximizes the total profit at the same time. In a later work they address the same problem for the 2-facility location problem with both simultaneous and sequential approaches, and conclude that the simultaneous approach leads to better results at an extremely high computational cost [37].

The undesirable effects of encroachment can also be captured from a different viewpoint in the sense that one can create multiple facility location models which are basically aimed at creating dispersed solutions by maximizing the dispersion among either the newly added units, or among the existing units and the new ones, which was first discussed in [77, 78]. These models are called the family of dispersion problems. The classical dispersion models try to maximize dispersion as a function of the distance/dissimilarity between the entities in the network.

Erkut and Neuman [33] propose four different types of the dispersion models based on different dispersion metrics. The first one is the MaxMinMin problem which maximizes the minimum distance between each pair of the selected facilities. The second one is the MaxSumMin which seeks to maximize the sum of the minimum distances from each facility to its closest neighbor. The third formulation is called MaxMinSum which takes the sum of the distances from each facility to all its neighbors and maximizes the smallest sum of the distances [78]. Finally the fourth formulation corresponds to the MaxSumSum which aims at maximizing the sum of all the hub distances for all the located facilities [77].

Saboonchi et al. [77, 78] address the franchise cannibalization issue for the first time through MaxMinSum and MaxSumSum dispersion problems. They developed several heuristics based on the Variable Neighborhood Search metaheuristic framework including various greedy constructive procedures and different shaking strategies. They finally discuss the tradeoffs among different solution strategies and the comparison of a results with that of the state-of-the-art heuristics demonstrate the high performance of their approaches.

The MaxMinMin problem on a general network is NP-complete which can be shown by reducing the original problem to a clique problem on a graph [32]. Several exact and heuristic solution procedures have been discussed in terms of their relationships with known graph theory problems. Erkut [32] demonstrate that this problem is also related to the maximum

clique (a set of mutually adjacent vertices) and the maximal independent set (a set of mutually nonadjacent vertices) problems. In a later work Erkut et al. [35] compare 10 heuristic solution procedures classified in four categories of Construction algorithms, Neighborhood algorithms, Projection algorithm and Interchange algorithms. Finally they show that the neighborhood and interchange heuristics provide better results than the construction heuristics.

Resende et al. [74] propose a heuristic method based on the GRASP and path relinking methodologies. Their experiments on three different sets of test instances, Glover, Geo and Ran (to be discussed later in Section 4.4) indicate that the proposed hybrid implementations compare favorably to previous metaheuristics, such as tabu search and simulated annealing.

Della Croce et al. [20] discuss a heuristic approach which relies on the equivalence of the MaxMinMin problem and the classical maximum clique graph problem. The heuristic is based on a dichotomic search where at each iteration a clique decision problem has to be solved. They conduct their experiments on the same data instances as in [74] and obtain the previous best results or improve them with proof of optimality for some of the $n = 250$ (medium-sized) instances.

Most recently, Porumbel et al. [70] proposed a simple local search with add and drop operations based on tabu rules. Their tabu rule offers low iteration complexity and strong diversification qualities leading to establishing a new lower bound for one of the data instances already studied in [20, 74], yet even by allowing larger amounts of running time (higher number of iterations with no improvements) they do not reach *all* of the best bounds reported in [20] (112 over 120 instances).

In this work we capture the cannibalization issue based on the MaxMinMin dispersion criterion which is the first application of `VNS` to this variant of the dispersion family. Going back to the franchise location literature, this heuristic method corresponds to the simultaneous approach for a multiple facility franchise location problem. Due to performance comparison purposes with the known benchmark test problems in [20, 70, 74], we apply the dispersion concept only among the newly added units. Of course in real-world applications one can easily modify the proposed solution procedure in order to incorporate the dispersion

among the new units and the already existing units, by taking the location of the existing units as a fixed parameter fed into the solution procedure.

In Section 4.2 we describe the problem in graph theoretical terms followed by its mixed integer formulation. Section 4.3 presents a detailed explanation of the proposed `VNS` heuristic solution procedure for the MaxMinMin p-dispersion problem and then the computational experiments on the largest known benchmark problems are discussed with the proof of optimality for most of the largest test instances that have never been solved to optimality in the literature. Finally we conclude the paper by highlighting our contributions and suggestions for future research.

3.2 Problem Statement and Mathematical Formulation

This section expresses the MaxMinMin p-dispersion problem in graph theoretical terms and then presents its respective mathematical formulation. Let $V = \{v_i, \forall i = 1, \ldots, n\}$, be a set of n vertices (potential locations) and v_i representing each member of this set. Let E be the set of $\binom{n}{2}$ edges of an undirected and fully connected graph $G(V, E)$, with $d_e > 0$ representing the distance over each edge $e \in E$. The value p is an integer such that $3 \leq p \leq |V|$. We define S as any subset of p vertices such that $S \subseteq V, |S| = p$. The subset of the vertices not present in the current solution is defined as $\bar{S}$ such that $\bar{S} = V \setminus S, |\bar{S}| = m = n - p$. For easier presentation purposes throughout the paper the size of the vertices outside the solution set is represented by m.

Let $G(S, E(S))$ be the subgraph induced by any subset S, as a result the objective function value $f(S)$ is defined as the smallest distance among all the p selected vertices in S:

$$f(S) = \min_{e \in E(S)} \{d_e\}.$$

The MaxMinMin problem intends to find the optimal subgraph $G(S^*, E(S^*))$, where:

$$S^* = \arg\max_S f(S).$$

This problem could be modeled as the following 0-1 mixed integer program as suggested in [32]:

$$\begin{aligned} \max \quad & Z \\ s.t. \quad & Z \leq M(2 - x_i - x_j) + d(v_i, v_j) \quad 1 \leq i < j \leq n \\ & \sum_{i=1}^{n} x_i = p \\ & x_i = \{0, 1\} \qquad\qquad\qquad\qquad 1 \leq i \leq n, \end{aligned}$$

where x_i is a binary decision variable defining if vertex v_i is selected, $d(v_i, v_j)$ is the distance between any pair of the located facilities at locations i and j and M is a sufficiently large number which could be set as D_{max}, the largest distance among all the n vertices. In Section 4.4.1 an upper bounding technique in order to obtain tighter bounds is discussed. The distances between all the vertices are taken as an input and stored in an $n \times n$ upper-triangular matrix with $d(v_i, v_i) = 0$.

3.3 VNS for the MaxMinMin Problem

Variable Neighborhood Search (`VNS`) is a metaheuristic or framework for building heuristics which is based on the idea of a systematic change of the neighborhood in order to escape from the valleys surrounding local optima, followed by a local search to find improved solutions. This general method has been proposed by Mladenović and Hansen [64] and has proven to lead to very successful heuristics for solving large combinatorial programs with applications in location theory, cluster analysis and several other fields. For a recent survey of the theoretical developments and applications including several hundred references see [45, 48].

The Basic Variable Neighborhood Search method (`BVNS`) has already been applied to the heaviest k-subgraph problem, MaxMinSum and MaxSumSum dispersion problems [11, 77, 78] and has shown to be among the most efficient methods compared to other heuristics for diversity problems [11, 63].

In order to represent the solution at each step of the heuristic we use the data structure suggested in [11]. The solution is represented by an array of n indices corresponding to

each vertex or candidate location, where the first p elements correspond to the subset of the current solution S.

The solution space U is represented by the $\binom{n}{p}$ subsets of V with cardinality p. In order to apply `VNS` a metric function is defined to evaluate the distance between any two solutions S and S':

$$\delta(S, S') = \delta(S', S) = |S \setminus S'|.$$

Based on the metric distance function defined above, the neighborhood of size k of a solution S is defined as:

$$N_k(S) = \{S' \in U | \delta(S, S') = k\}; k = 1, 2, \ldots, \min\{p, m\}.$$

Throughout this paper the following notations are used:

- $S_{best/cur}$: the best/current solution set corresponding to the best/current objective function value;
- $f(S_{best/cur})$: the best/current objective function value that corresponds to the smallest distance among all the selected vertices in the best/current solution set $S_{best/cur}$;
- $\alpha(v_i)$: the *first* closest *distance* to each $v_i \in V$ from any $v_j \in S$;
- v_i^1: the closest *vertex* to each $v_i \in V$ from any $v_j \in S$;
- $\beta(v_i)$: the *second* closest *distance* to each $v_i \in V$ from any $v_j \in S$;
- v_i^2: the *second* closest *vertex* to each $v_i \in V$ from any $v_j \in S$;
- v_{exit}: the vertex inside the solution set that is a candidate to *leave* the solution set ($v_{exit} \in S$);
- v_{enter}: the vertex outside the solution set that is a candidate to *enter* the solution set ($v_{enter} \in \bar{S}$);
- S_{temp}: the temporary solution set after removing the v_{exit}, where $|S_{temp}| = p - 1$;
- $f(S_{temp})$: the smallest distance among the $(p-1)$ members of the temporary solution set S_{temp}.

In Algorithm 3.1 we define the `VNS` function and then in the following sections we explain in details the functions embedded in our general framework. The stopping criterion is the limit on the maximum number of iterations with no improvements it_{max} [70]. The already elapsed cumulative time and iteration in the overall procedure are noted by $t_{elapsed}$ and $it_{elapsed}$. The k_{min} and k_{step} (shake step size) parameters are set by default to 1, and the k_{max} (maximum shake size) is set to a coefficient of $\min\{p, m\}$ as discussed later.

```
function VNS (k_min, k_step, k_max)
S_cur ← Initialize();
S_best ← S_cur;
it_elapsed = 0;
while it_elapsed ≤ it_max do
    k_cur = k_min;
    while k_cur ≤ k_max and it_elapsed ≤ it_max do
        S_cur ← Shake(S_cur);
        S_cur ← LocalSearch(S_cur);
        S_cur ← RefinedLocalSearch(S_cur);
        if f(S_cur) > f(S_best) then
            S_best ← S_cur;
            k_cur = k_min;
            it_elapsed = 0;
        else
            k_cur = k_cur + k_step;
            it_elapsed + +;
        end
    end
end
```

Figure 3.1: Pseudo Code for the VNS Framework

3.3.1 Initialization

The initial solution could be created at random or in a greedy manner. Based on the *Random add* method the initial solution is simply created by choosing p vertices at random. The *Greedy add* and the *Greedy drop* construction heuristics could have been applied as suggested in [77, 78]. The use of greedy methods in order to create the initial solution is not necessary in our method, since based on a series of preliminary comparisons of the various starting methods we observed that the random start method leads to high quality and robust solutions. As a result we decided to use a simple random initialization to obtain

more diversified solutions in repeated runs and of course the efficient`LocalSearch` procedure would soon find high quality solutions.

3.3.2 Local Search

After having created the initial solution, the `LocalSearch` procedure is implemented by performing 1-interchange swaps on the current solution as shown in Algorithm 3.2. This means that at each iteration only *one* vertex is swapped at a time. The swap could be done whenever the first (first improvement strategy) or the best (best improvement strategy) contribution is made to the current objective value. In this work the first improvement strategy is implemented in order to create more diversification considering the flat landscape of the objective function in the MaxMinMin problem. Besides the first improvement strategy has much less iteration complexity in practice.

In order to start the `LocalSearch` procedure the gain obtained from swapping the selected entering candidate with the selected leaving candidate should be evaluated as explained in Section 4.3.3.1. The two main `Contribution` and `Update` functions will be explained in details in the following subsections.

```
function LocalSearch(S)
repeat
    (v_exit, v_enter, gain) ← Contribution(S);
    if gain > 0 then
        Swap(v_exit, v_enter);
        Update(v_exit, v_enter, S);
    end
until gain > 0;
```

Figure 3.2: Pseudo Code for the Local Search Procedure

3.3.2.1 Contribution

In the proposed `LocalSearch` procedure the `Contribution` function illustrated in Algorithm 3.3 can determine the first or best swap candidates $v_{exit} \in S$ and $v_{enter} \in \bar{S}$, as well as its corresponding contribution to the current objective function value.

For a more efficient illustration in Algorithm 3.3 we have defined the *position(i,set)* function which returns the vertex in the i^{th} position of the subsets S or $\bar{S}$. In order to initiate the `Contribution` function a leaving candidate $v_{exit} = position(i, S)$ is chosen. A swap with any entering candidate $v_{enter} = position(j, \bar{S})$ will make an improvement only in the following two cases:

- If the selected $v_{exit} \in S$ to be removed from the solution set corresponds to v^1_{enter} for the entering candidate $v_{enter} \in \bar{S}$, the $\alpha(v_{enter})$ value would be also removed. In this case the gain would be positive only if the $\beta(v_{enter})$ value is greater than the current solution $f(S_{cur})$, and thus the gain would be:

$$gain \leftarrow min\{f(S_{temp}), \beta(v_{enter})\} - f(S_{cur}).$$

- If the selected $v_{exit} \in S$ does NOT correspond to the v^1_{enter} for the $v_{enter} \in \bar{S}$, the gain would be positive only if the $\alpha(v_{enter})$ value is greater than the current solution $f(S_{cur})$, and thus the gain would be:

$$gain \leftarrow min\{f(S_{temp}), \alpha(v_{enter})\} - f(S_{cur}).$$

Apart from the above mentioned situations the swap will not make an improvement, and thus the algorithm will move on to the next swap candidates. Calculating the $f(S_{temp})$ after having removed the v_{exit} requires $O(p)$ time using the α and β values, and with the first improvement strategy the `Contribution` function stops as soon as an improving solution is found. As a result in the worst case it is implemented in $O(pm)$ (the maximum of and $O(p^2)$ and $O(pm)$) time per `LocalSearch` iteration. It should be noted that for all the datasets studied in this work the following relationship exists: $O(p) < O(m) < O(n) < O(p^2) < O(pm)$.

The $O(pm)$ worst case complexity reaches its maximum value when $p = n/2$, as a result the maximum complexity is $O(n^2/4)$ which is already inferior that $O(n^2)$ and of course in our experiments such a situation would not happen since $p < n/2$. Besides, extensive empirical results demonstrate that owing to the efficient use of the α and β values the average complexity is much lower and the worst case complexity is almost never attained.

```
function Contribution(S)
gain = 0; i = 0;
while i ≤ p and gain = 0 do
    v_exit ← position(i, S);
    remove v_exit and find f(S_temp);
    j = 0;
    while j ≤ m and gain = 0 do
        v_enter ← position(j, S̄);
        if v_exit = v^1_enter then
            if β(v_enter) > f(S_cur) then
                gain ← min{f(S_temp), β(v_enter)} − f(S_cur);
            end
        else
            if α(v_enter) > f(S_cur) then
                gain ← min{f(S_temp), α(v_enter)} − f(S_cur);
            end
        end
        j++ //move on to the next position in S̄//;
    end
    i++ //move on to the next position in S//;
end
if gain > 0 then
    return (v_exit, v_enter, gain);
end
```

Figure 3.3: Pseudo Code for Determining the First Improving Swap and its Contribution (First Improvement Strategy)

3.3.2.2 Update

The `Update` procedure demonstrated in Algorithm 3.4 performs all the required updates before proceeding to the subsequent iteration. It is implemented in three different phases in order to update the objective function value $f(S_{cur})$, the $\alpha(v_i)$ and $\beta(v_i)$ values for all the $v_i \in V$.

The update of the objective function value $f(S_{cur})$ is straightforward and is computed in $O(p)$ total time by simply finding the smallest updated $\alpha(v_i)$ value for all $v_i \in S$. For the update of the $\alpha(v_i)$ and $\beta(v_i)$ values for all $v_i \in V$ three cases might occur:

- **Case 1:** If the selected leaving candidate $v_{exit} \in S$ is neither the first, nor the second closest vertex to the $v_i \in V$, then depending on the relative values of the $\alpha(v_i)$, $\beta(v_i)$

and its distance to the entering candidate which has just entered the solution set, $d(v_i, v_{enter})$, the update of the new $\alpha(v_i)$ and $\beta(v_i)$ would be done in $O(1)$ time:

  - If the $d(v_i, v_{enter}) \leq \alpha(v_i)$, the old $\alpha(v_i)$ value would replace the $\beta(v_i)$, and the $d(v_i, v_{enter})$ would replace $\alpha(v_i)$.
  - If the $\alpha(v_i) < d(v_i, v_{enter}) \leq \beta(v_i)$, the old $\alpha(v_i)$ value would not change and the $d(v_i, v_{enter})$ would replace $\beta(v_i)$.
  - If the $d(v_i, v_{enter}) > \beta(v_i)$, neither $\alpha(v_i)$ nor $\beta(v_i)$ would change.

- **Case 2:** If the selected leaving candidate $v_{exit} \in S$ to be removed corresponds to the *first* closest vertex v_i^1 for $v_i \in V$, then the old $\alpha(v_i)$ value should be removed. As a result only the $\beta(v_i)$ and $d(v_i, v_{enter})$ values need to be compared:
  - If the $d(v_i, v_{enter}) \leq \beta(v_i)$, then the $\beta(v_i)$ would not change and the $d(v_i, v_{enter})$ would replace $\alpha(v_i)$.
  - If the $d(v_i, v_{enter}) > \beta(v_i)$, the old $\beta(v_i)$ value would replace $\alpha(v_i)$ and the new updated $\beta(v_i)$ should be recomputed in $O(p)$ time.

- **Case 3:** If the selected leaving candidate $v_{exit} \in S$ to be removed corresponds to the *second* closest vertex v_i^2 for $v_i \in V$ then the old $\beta(v_i)$ value should be removed. As a result the update would be:
  - If the $d(v_i, v_{enter}) \leq \alpha(v_i)$, the old $\alpha(v_i)$ would replace as the new updated $\beta(v_i)$ and the $d(v_i, v_{enter})$ would replace $\alpha(v_i)$.
  - If the $\alpha(v_i) < d(v_i, v_{enter}) \leq \beta(v_i)$, the $\alpha(v_i)$ value would not change and the $d(v_i, v_{enter})$ would replace $\beta(v_i)$.
  - If the $d(v_i, v_{enter}) > \beta(v_i)$, the old $\alpha(v_i)$ value would not change and the new updated $\beta(v_i)$ should be recomputed in $O(p)$ time.

Hence, the proposed `Update` procedure is very efficient since the update of $\alpha(v_i)$ and $\beta(v_i)$ values is done in $O(1)$ time, except for only *two* possible cases where the $\beta(v_i)$ should be recomputed in $O(p)$ time. This approach allows us carry forward the information gathered in one iteration to the subsequent ones and to do the updates at a low linear complexity.

```
function Update(v_exit, v_enter, S)
Find the updated f(S_cur) (the smallest updated α(v_i) for all the v_i ∈ S);
forall the v_i ∈ V do
    if v_exit ≠ v_i^1 ≠ v_i^2 //Case 1// then
        if d(v_i, v_enter) ≤ α(v_i) then
            β(v_i) ← α(v_i);
            α(v_i) ← d(v_i, v_enter);
        else
            if α(v_i) < d(v_i, v_enter) ≤ β(v_i) then
                β(v_i) ← d(v_i, v_enter);
            end
        end
    else
        if v_exit = v_i^1 //Case 2// then
            if d(v_i, v_enter) ≤ β(v_i) then
                α(v_i) ← d(v_i, v_enter);
            else
                α(v_i) ← β(v_i);
                β(v_i) ← recompute;
            end
        else
            if v_exit = v_i^2 //Case 3// then
                if d(v_i, v_enter) ≤ α(v_i) then
                    β(v_i) ← α(v_i);
                    α(v_i) ← d(v_i, v_enter);
                else
                    if α(v_i) < d(v_i, v_enter) ≤ β(v_i) then
                        β(v_i) ← d(v_i, v_enter);
                    else
                        β(v_i) ← recompute;
                    end
                end
            end
        end
    end
end
```

Figure 3.4: Pseudo Code for the Update Procedure

3.3.3 Refined Local Search

The `RefinedLocalSearch` is called right after the `LocalSearch` function in the general `VNS` framework illustrated in Algorithm 3.1. The existence of plateaus or the flat landscapes of the MaxMinMin problem makes it more difficult for the ascent procedures to find an improvement in the objective function especially as the size and the density of the graphs increase [11, 20, 74]. Resende et al. [74] present a new notion of improvement: a swap will be accepted not only if it improves the solution value, but also if it creates a solution subset S with a lower number of edges equal to the binding objective function value. This has inspired us to develop the `RefinedLocalSearch` plateau search mechanism within the `VNS` metaheuristic framework.

The `RefinedLocalSearch` procedure is efficient both in the presence, and the absence of plateaus. The reason is that it makes all the possible swaps that will improve the quality of the edges in the solution set that are not necessarily binding. As a result in the presence of plateaus it will decrease the thickness of the plateau, and in the absence of plateaus it will exchange the low quality edges with the ones of higher quality which will facilitate further improvements by the `LocalSearch` procedure in the subsequent iterations. Of course this procedure could even lead to improved objective values if the whole plateau is removed.

In order to choose the leaving candidate v_{exit}, the `RefinedLocalSearch` procedure starts by defining a smaller subset of the solution set which consists of only the vertices in the solution set whose α values are inferior than a certain value. The tolerance is a parameter that is tuned empirically. In order to do so the θ parameter is introduced and by changing the its value one can increase or decrease the size of the above explained subset. Here D_{max} is the largest distance among all the vertices and of course we can include the whole solution set by setting $\theta = 1$.

The entering candidate would be any vertex whose second closest distance $\beta(v_k)$ is greater than the current solution $f(S_{cur})$ and whose closest vertex v_k^1 is a member of the subset that has already been formed. Such a candidate vertex would be swapped with its closest vertex v_k^1 which is already a member of the subset. This method ensures that after the swap the

current solution would never deteriorate and the swaps are done until eligible candidates are found.

```
function RefinedLocalSearch(S)
repeat
    tolerance = θ * (D_max − f(S_cur));
    Set = {v_i ∈ S|α(v_i) ≤ f(S_cur) + tolerance};
    k = 0;
    bool = false;
    while k ≤ m and bool = false do
        v_enter ← position(k, S̄);
        if β(v_enter) ≥ f(S_cur) then
            if v^1_enter ∈ Set then
                v_exit ← v^1_enter;
                Swap(v_exit, v_enter);
                Update(v_exit, v_enter, S);
                bool = true;
            end
        else
            k++ //move on to the next position in S̄//;
        end
    end
until bool=true;
```

Figure 3.5: Pseudo Code for the Refined Local Search Procedure

3.3.4 Shake

The perturbation in most VNS-based heuristics is done in a simple manner by choosing a random vertex from the k_{th} neighborhood, i.e. $N_k(S)$ from the current solution S and then repeating k times the random swap move. The RandomShake function does so by choosing one random leaving and entering candidate at each iteration with updates in between each swap. However, we have developed two additional shake functions in order to control the perturbation operation in a more intelligent manner.

The SemiGreedyShake function fixes a random leaving candidate from the current solution set S and then chooses an entering candidate whose smallest distance to the remaining solution vertices in the solution set, i.e. the $\alpha(v_i)$ value (or the $\beta(v_i)$ value if the selected leaving candidate corresponds to its v_i^1) is the largest. As a result this shake method does not guarantee that a deterioration in the objective function value would not occur, yet it

simply chooses a reasonable entering candidate after having fixed the leaving candidate. This procedure is repeated until the shake size of k is attained. Each iteration is performed in $O(m)$ time and after each swap the `Update` function is called.

In order to have a more intensified shake operation the `GreedyShake` function could be used which works the same way as the `SemiGreedyShake` function in order to find the entering candidates. Yet, the leaving candidates would be selected only among the vertices that form the binding distances (the objective function value) within the solution set.

We have designed a series of preliminary experiments as suggested in [77, 78] and found out empirically that the `SemiGreedyShake` function is superior than the other two shaking options for the MaxMinMin problem. Therefore, only this shake function would be used in the final experiments in Section 4.4 where it is demonstrated that the more intelligent shake functions dominate the purely random ones.

3.4 Computational Experiments

In this section we have selected two of the largest *Ran* and *Geo* benchmark instances that have been used for comparison purposes for the MaxMinMin problem leading to 80 instances in total [20, 70, 74]. Here we do not consider the smaller instances with $n = 100$ previously used in the literature as they have been already solved to optimality and do not impose any challenge to our proposed method. A brief description of the characteristics of the datasets is given below:

- Geo: this dataset consists of 40 matrices where the distances are generated by first randomly forming n points with a random number of coordinates between 2 and 21 and then with random coordinates in the range $[0, 100]$. The size of the instances is $n = 250$ and 500 with $p = 0.1n$ and $0.3n$ respectively.
- Ran: this dataset consists of 40 matrices with integer numbers randomly generated within the interval $[50, 100]$ for all the instances except for the instances with $n = 500$ and $p = 0.3n$, where the distances are generated in the interval $[1, 200]$.

First we describe our experiments that were designed to study the performance of different settings within the `VNS` framework and then analyze the overall results obtained by different

methods over all the test problems. We then compare a results with that of the state-of-the-art heuristics in the literature and finally find the optimal solution for most of the large instances using exact methods.

3.4.1 Experiments Setup

The proposed `VNS` implementation allows various settings and methods within its framework. In order to initialize the `VNS` three different methods have been discussed: Random add, Greedy add and Greedy drop. There are also three different shaking possibilities: Random shake, Semi-Greedy shake and Greedy shake. On the other hand at each iteration of `VNS` either an improvement in the solution is made or not. In case of no improvement a decision on how to start the next iteration should be made. The next iteration is either started form the already best solution obtained, or from the current solution just obtained. The former will lead to more intensification in the search, whereas the latter favors diversification. Besides, the `LocalSearch` procedure can pursue a first improvement strategy favoring more diversification, versus best improvement strategy leading to more intensification.

There also are two parameters to be tuned in the general framework: 1) the θ parameter in Algorithm 3.5 which adjusts the size of the subset to choose the leaving candidates, and 2) the k_{max} whose value by default is $\min\{p, m\}$ which might be a large value depending on the instance size. In the experiments setup phase the tested values were $\theta = 0.03, 0.06$ and 0.1 and also $k_{max} = 0.5 * \min\{p, m\}, 0.75 * \min\{p, m\}$ and $min\{p, m\}$.

As mentioned in Section 4.3 we conducted preliminary experiments to choose the most promising setting for the final experiments (for more details see e.g. [77, 78]). Empirically, the following setting has been selected for the MaxMinMin problem:

- The current iteration and first improvement `LocalSearch` strategies lead to overall better results and lower complexity compared to other general frameworks.
- The Greedy add and drop initialization methods do not have significant differences and lead to the same results in repeated runs; as a result the heuristic is started randomly for more diversification.

- The Semi-Greedy Shake method leads to a significantly higher quality over all data instances.
- There is not a definitive winner among the different parameter combinations; yet empirically $\theta = 0.1$ and $k_{max} = 0.75 * \min\{p, m\}$ seem to lead to relatively higher quality solutions over all the 80 data instances.

The heuristics are coded in C++ and compiled with the $-O2$ optimization option, and are run 30 times on a linux machine with an intel compiler of 2.667 GHz and 3Gb Ram. The stopping criteria is the maximum number of iterations with no improvements which is empirically set to $it_{max} = k(n^2/p)$. We have run two series of tests with different stopping criteria with $k = 1$ and 2.5 for the smaller $n = 250$ instances, and $k = 10$ and 25 for the $n = 500$ instances. The best obtained results for all the 80 data instances are presented under the "Our best" columns of Tables 3.III and 3.IV , where the values in bold show that best ever results have been obtained.

3.4.2 Results and Analysis

As mentioned earlier two series of tests are performed with different stopping criteria. Each instance is run 30 rimes and in order to have a better idea of the quality and the robustness of a results the "Coefficient of variation" is reported for each instance group calculated as follows:

$$\text{CV} = \frac{\text{standard deviation}}{\text{mean}} \times 100.$$

Tables 3.I and 3.II represent the summary of a results for the stopping criteria of $it_{max} = k(n^2/p)$ with smaller and larger k values respectively. The first column shows the name of the instances separating the instances based on type and size, the second column represents the value of the k coefficient adjusting the stopping criteria and the third column shows the number of instances for which the best ever solution has been found (20 instances in each group). The last three columns demonstrate the quality of our solutions in terms of the coefficient of variation (CV) averaged over all the 20 instances within each dataset, the average time when the best solution (the best solution for each specific run and not

necessarily the best *ever* solution) was found (Time-best) and the average running time (Time-ave) of the 30 runs for each group.

Both tests series (with shorter and longer stopping criteria) obtain *all* of the 80 best known results in multiple runs. Comparison of the two tables shows that allowing larger number of iterations with no improvements increases the total average running time (121.77 vs. 302.51), and decreases the CV or increases the quality of the solutions (0.08 vs. 0.07), i.e. allowing approximately 59.75% longer running times on average have led to an improvement of 12.5% in the average CV. Here examining longer running times (maximum number of iterations with no improvements) would not make any sense, since even with the shorter running times all the 80 best results have been obtained and the interpretation of the tradeoff between running times and solution quality is a matter of the decision maker's choice. Based on the same logic trying even shorter running times is not necessary either, as the average times are already very low and this would only increase the average CV values.

The comparison of a results with that of the state-of-the-art heuristics is demonstrated in Tables 3.III and 3.IV where the first three columns represent the name of each instance and its size. In the columns RMGD, DCGL and MMDP (names as suggested in [70]), the best results obtained by Resende et al. [74], Della Croce et al. [20] and Porumbel et al. [70] are reported respectively. These methods are among the most powerful heuristics ever applied on the datasets. The last column summarizes our results where the "Our best" column reports the best results obtained in 30 runs of our heuristic. The results highlighted in bold show that our method finds *all* the best solutions ever obtained in the literature including the new improvement reported in [70], among which 53 out of 80 are proven to be optimal in Tables 3.V and 3.VI . It should be noted that none of the previous methods have succeeded in finding all of the best benchmark results.

Table 3.I : Summary of Results for Shorter Running Times

Instance	k	# Best	CV	Time-best	Time-ave
Geo 250	1	20	0.04	0.65	3.04
Ran 250	1	20	0.03	0.19	1.8
Geo 500	10	20	0.05	42.93	320.5
Ran 500	10	20	0.2	8.84	161.73
All 80 instances		**80**	**0.08**	**13.15**	**121.77**

Table 3.II : Summary of Results for Longer Running Times

Instance	k	# Best	CV	Time-best	Time-ave
Geo 250	2.5	20	0.01	0.89	6.85
Ran 250	2.5	20	0.04	0.21	3.91
Geo 500	25	20	0.03	112.01	805.77
Ran 500	25	20	0.19	46.91	393.51
All 80 instances		**80**	**0.07**	**40.01**	**302.51**

The performance of our method is clearly superior than the several methods tested in [74] (Simulated annealing, Tabu Search, GRASP and path-relinking etc) both in terms of the solution quality (69 better solutions out of 80) and time (hundreds of seconds for RMGD). The moderate and high execution times over the larger instances reported by MMDP in [70] are obtained by a limit of $10000n$ unsuccessful iterations leading to 57 over 80 best solutions (0.24% deviation and 40.6 average time), and also the limit of $2000000n$ iterations with no improvements leading to 72 over 80 best results (0.06% deviation and 7840.75 average time). As a result the superiority of our method in terms of the solution quality and computational times is evident. Besides the algorithm proposed in [70] is launched from an initial solution already created by a constructive heuristic which seems not to be included in the total running times. Furthermore, the Time-best column in Table 3.I demonstrates that our best solutions were obtained in an average of 13.15 seconds where the $n = 250$ *optimal* values were obtained in an average of only 0.42 seconds. There are no detailed computational times reported for each individual group of instances for DCGL [20]. The DCGL takes a complex and refined internal max-clique heuristic already in literature in each iteration of its dichotomic search and it is not clear if the stopping criterion of the fixed 250000 total iterations corresponds to this internal heuristic or the overall MaxMinMin heuristic. Yet, their computational times seem to be of tens of seconds [70].

3.4.3 Comparison With Exact Methods

In order to verify the quality of the solutions obtained by our heuristics we used the Ilog cplex 12.4 for exact solutions. Yet, even the smallest instances seemd impossible for exact methods to solve in a reasonable time. As a result we provided Ilog cplex with the best solution we ever obtained from our heuristics in Section 4.4.1 as an initial solution and also calculated an exact upper bound in order to facilitate the problem resolution.

Table 3.III : Comparison of the Best Known Results for the Geo Instances

Instance	n	p	RMGD	DCGL	MMDP	Our best
Geo 250 1	250	25	**171.01**	**171.01**	**171.01**	**171.01**
Geo 250 2	250	25	19.7	**20.03**	20	**20.03**
Geo 250 3	250	25	**137.75**	**137.75**	**137.75**	**137.75**
Geo 250 4	250	25	171.21	**171.96**	**171.96**	**171.96**
Geo 250 5	250	25	148.72	**148.76**	**148.76**	**148.76**
Geo 250 6	250	25	99.8	**100.38**	**100.38**	**100.38**
Geo 250 7	250	25	**175.71**	**175.71**	**175.71**	**175.71**
Geo 250 8	250	25	164.03	**164.51**	**164.51**	**164.51**
Geo 250 9	250	25	179.16	**179.18**	**179.18**	**179.18**
Geo 250 10	250	25	102.19	**102.23**	**102.23**	**102.23**
Geo 250 11	250	75	31.37	**31.72**	31.37	**31.72**
Geo 250 12	250	75	93.84	**93.91**	**93.91**	**93.91**
Geo 250 13	250	75	145.06	**145.22**	145.13	**145.22**
Geo 250 14	250	75	70.12	**70.7**	70.32	**70.7**
Geo 250 15	250	75	142.5	**142.54**	**142.54**	**142.54**
Geo 250 16	250	75	**108.05**	**108.05**	**108.05**	**108.05**
Geo 250 17	250	75	124.25	**124.63**	**124.63**	**124.63**
Geo 250 18	250	75	148.63	**148.76**	**148.76**	**148.76**
Geo 250 19	250	75	133.87	**134.74**	**134.74**	**134.74**
Geo 250 20	250	75	**147.83**	**147.83**	**147.83**	**147.83**
Geo 500 1	500	50	123.74	**124.79**	**124.79**	**124.79**
Geo 500 2	500	50	13.4	**13.79**	**13.79**	**13.79**
Geo 500 3	500	50	164.13	**165.04**	**165.04**	**165.04**
Geo 500 4	500	50	131.62	**132.5**	**132.5**	**132.5**
Geo 500 5	500	50	28.07	**28.55**	28.18	**28.55**
Geo 500 6	500	50	27.8	**28.6**	28.18	**28.6**
Geo 500 7	500	50	131.34	**132.51**	**132.51**	**132.51**
Geo 500 8	500	50	112.68	**113.6**	**113.6**	**113.6**
Geo 500 9	500	50	168.24	**168.96**	**168.96**	**168.96**
Geo 500 10	500	50	159.68	**159.98**	**159.98**	**159.98**
Geo 500 11	500	150	93.49	**93.97**	93.66	**93.97**
Geo 500 12	500	150	71.12	**71.46**	**71.46**	**71.46**
Geo 500 13	500	150	133.99	**134.47**	**134.47**	**134.47**
Geo 500 14	500	150	111.04	**111.63**	**111.63**	**111.63**
Geo 500 15	500	150	35.71	**36.18**	**36.18**	**36.18**
Geo 500 16	500	150	132.43	**132.58**	**132.58**	**132.58**
Geo 500 17	500	150	129.04	**129.49**	**129.49**	**129.49**
Geo 500 18	500	150	71.85	**72.65**	**72.65**	**72.65**
Geo 500 19	500	150	123.95	**123.99**	**123.99**	**123.99**
Geo 500 20	500	150	123.14	**123.43**	**123.43**	**123.43**

Table 3.IV : Comparison of the Best Known Results for the Ran Instances

Instance	n	p	RMGD	DCGL	MMDP	Our best
Ran 250 1	250	25	59	**61**	**61**	**61**
Ran 250 2	250	25	60	**61**	**61**	**61**
Ran 250 3	250	25	60	**61**	**61**	**61**
Ran 250 4	250	25	59	**61**	**61**	**61**
Ran 250 5	250	25	60	**61**	**61**	**61**
Ran 250 6	250	25	60	**61**	**61**	**61**
Ran 250 7	250	25	60	**61**	**61**	**61**
Ran 250 8	250	25	60	**61**	**61**	**61**
Ran 250 9	250	25	60	**61**	**61**	**61**
Ran 250 10	250	25	60	**61**	**61**	**61**
Ran 250 11	250	75	**52**	**52**	**52**	**52**
Ran 250 12	250	75	51	**52**	**52**	**52**
Ran 250 13	250	75	**52**	**52**	**52**	**52**
Ran 250 14	250	75	**52**	**52**	**52**	**52**
Ran 250 15	250	75	51	**52**	**52**	**52**
Ran 250 16	250	75	**52**	**52**	**52**	**52**
Ran 250 17	250	75	**52**	**52**	**52**	**52**
Ran 250 18	250	75	**52**	**52**	**52**	**52**
Ran 250 19	250	75	**52**	**52**	**52**	**52**
Ran 250 20	250	75	51	**52**	**52**	**52**
Ran 500 1	500	50	54	**55**	**55**	**55**
Ran 500 2	500	50	55	**56**	**56**	**56**
Ran 500 3	500	50	55	55	**56**	**56**
Ran 500 4	500	50	55	**56**	**56**	**56**
Ran 500 5	500	50	54	**56**	**56**	**56**
Ran 500 6	500	50	54	**55**	**55**	**55**
Ran 500 7	500	50	54	**56**	**56**	**56**
Ran 500 8	500	50	55	**56**	55	**56**
Ran 500 9	500	50	55	**56**	**56**	**56**
Ran 500 10	500	50	54	**56**	**56**	**56**
Ran 500 11	500	150	4	**5**	**5**	**5**
Ran 500 12	500	150	4	**5**	**5**	**5**
Ran 500 13	500	150	4	**5**	**5**	**5**
Ran 500 14	500	150	4	**5**	**5**	**5**
Ran 500 15	500	150	4	**5**	**5**	**5**
Ran 500 16	500	150	4	**5**	**5**	**5**
Ran 500 17	500	150	4	**5**	**5**	**5**
Ran 500 18	500	150	4	**5**	**5**	**5**
Ran 500 19	500	150	4	**5**	**5**	**5**
Ran 500 20	500	150	4	**5**	**5**	**5**

Table 3.V : Optimality Check for the Geo Instances

Instance	n	p	LB	UB	Dmax	Cplex	Gap	Time
Geo 250 1	250	25	**171.01**	220	267.19	171.01	**optimal**	1023
Geo 250 2	250	25	**20.03**	98	130.21	20.03	**optimal**	22
Geo 250 3	250	25	**137.75**	189	231.83	137.75	**optimal**	317
Geo 250 4	250	25	**171.96**	222	269.26	171.96	**optimal**	425
Geo 250 5	250	25	**148.76**	199	261.61	148.76	**optimal**	291
Geo 250 6	250	25	**100.38**	156	197.9	100.38	**optimal**	244
Geo 250 7	250	25	**175.71**	226	296.89	175.71	**optimal**	334
Geo 250 8	250	25	**164.51**	215	258.77	164.51	**optimal**	363
Geo 250 9	250	25	**179.18**	228	289.15	179.18	**optimal**	856
Geo 250 10	250	25	**102.23**	158	207.63	102.23	**optimal**	183
Geo 250 11	250	75	**31.72**	96	162.88	31.72	**optimal**	14
Geo 250 12	250	75	**93.91**	150	241.24	93.91	**optimal**	41
Geo 250 13	250	75	**145.22**	199	269.78	145.22	**optimal**	141
Geo 250 14	250	75	**70.7**	131	193.65	70.7	**optimal**	24
Geo 250 15	250	75	**142.54**	194	275.29	142.54	**optimal**	40
Geo 250 16	250	75	**108.05**	163	242.16	108.05	**optimal**	101
Geo 250 17	250	75	**124.63**	178	250.58	124.63	**optimal**	82
Geo 250 18	250	75	**148.76**	201	273.66	148.76	**optimal**	134
Geo 250 19	250	75	**134.74**	189	264.01	134.74	**optimal**	80
Geo 250 20	250	75	**147.83**	201	274.92	147.83	**optimal**	69
Geo 500 1	500	50	124.79	185	247.02	146.71	17.57%	1209600
Geo 500 2	500	50	**13.79**	97	135.29	13.79	**optimal**	684
Geo 500 3	500	50	165.04	223	280.72	189.31	14.71%	1209600
Geo 500 4	500	50	132.5	192	255.87	152.33	14.97%	1209600
Geo 500 5	500	50	**28.55**	111	158.58	28.55	**optimal**	4053
Geo 500 6	500	50	**28.6**	112	156.96	28.6	**optimal**	9111
Geo 500 7	500	50	132.51	193	247.82	152.02	14.72%	1209600
Geo 500 8	500	50	113.6	177	225.84	134.35	18.27%	1209600
Geo 500 9	500	50	168.96	226	278.94	202.07	19.60%	1209600
Geo 500 10	500	50	159.98	219	280.9	192.8	20.52%	1209600
Geo 500 11	500	150	**93.97**	157	236.76	93.97	**optimal**	4136
Geo 500 12	500	150	**71.46**	137	226.01	71.46	**optimal**	370
Geo 500 13	500	150	**134.47**	194	279.81	134.47	**optimal**	13804
Geo 500 14	500	150	**111.63**	173	247.57	111.63	**optimal**	3043
Geo 500 15	500	150	**36.18**	106	181.86	36.18	**optimal**	120
Geo 500 16	500	150	**132.58**	193	274.5	132.58	**optimal**	23440
Geo 500 17	500	150	**129.49**	190	274.14	129.49	**optimal**	3824
Geo 500 18	500	150	**72.65**	139	217.5	72.65	**optimal**	2859
Geo 500 19	500	150	**123.99**	184	259.67	123.99	**optimal**	5245
Geo 500 20	500	150	**123.43**	183	262.88	123.43	**optimal**	13871

The upper bound is calculated as follows: for each of the n vertices their distances to their furthest $(p-1)$ vertices are listed in decreasing order. As a result for each vertex there is a value that represents its distance to its $(p-1)_{th}$ furthest vertex. Now if the above n values are sorted in decreasing order, it is assured that the optimal solution can never exceed the value in the p_{th} rank in this sorted list.

After having provided the initial lower bound and the upper bound, we ran ILOG CPLEX for all the instances for as long as possible considering the available computational resources. In Tables 3.V and 3.VI the "Lower bound (LB)" column represents our best results already found, the "Upper Bound (UB)" column addresses the above explained exact upper bounds and the D_{max} is the maximum distance among all the n vertices for each instance. The last three columns represent the best bound found by ILOG CPLEX, its optimality status and the time in seconds for a single run of ILOG CPLEX to find the optimal solution or to terminate in general due to limited computational resources. The results prove the optimality of 33 solutions and seven tighter upper bounds for the Geo instances. Here the maximum running time is set to two weeks (336 hours). The Ran instances require more computational resources due to the existence of thick plateaus in their data structure. As a result we proved the optimality of all the $n = 250$ Ran instances and found tighter bounds for $n = 500$ ones. The maximum running time for the Ran instances is three days (72 hours) due to the high computational resources requirement of such instances and the fact that longer running times lead to very negligible improvements in the Gap.

Table 3.VI : Optimality Check for the Ran Instances

Instance	n	p	LB	UB	Dmax	Cplex	Gap	Time
Ran 250 1	250	25	**61**	97	100	**61**	**optimal**	21226
Ran 250 2	250	25	**61**	97	100	**61**	**optimal**	19125
Ran 250 3	250	25	**61**	97	100	**61**	**optimal**	27730
Ran 250 4	250	25	**61**	97	100	**61**	**optimal**	24031
Ran 250 5	250	25	**61**	97	100	**61**	**optimal**	31043
Ran 250 6	250	25	**61**	97	100	**61**	**optimal**	22673
Ran 250 7	250	25	**61**	97	100	**61**	**optimal**	19190
Ran 250 8	250	25	**61**	97	100	**61**	**optimal**	25624
Ran 250 9	250	25	**61**	97	100	**61**	**optimal**	20037
Ran 250 10	250	25	**61**	97	100	**61**	**optimal**	17230
Ran 250 11	250	75	**52**	86	100	**52**	**optimal**	725
Ran 250 12	250	75	**52**	86	100	**52**	**optimal**	793
Ran 250 13	250	75	**52**	86	100	**52**	**optimal**	745
Ran 250 14	250	75	**52**	86	100	**52**	**optimal**	402
Ran 250 15	250	75	**52**	86	100	**52**	**optimal**	600
Ran 250 16	250	75	**52**	86	100	**52**	**optimal**	428
Ran 250 17	250	75	**52**	86	100	**52**	**optimal**	1414
Ran 250 18	250	75	**52**	86	100	**52**	**optimal**	392
Ran 250 19	250	75	**52**	86	100	**52**	**optimal**	1180
Ran 250 20	250	75	**52**	86	100	**52**	**optimal**	1767
Ran 500 1	500	50	55	96	100	91	65%	259200
Ran 500 2	500	50	56	96	100	91	62.5%	259200
Ran 500 3	500	50	56	96	100	92	64%	259200
Ran 500 4	500	50	56	96	100	92	64%	259200
Ran 500 5	500	50	56	96	100	91	62.5%	259200
Ran 500 6	500	50	55	96	100	91	65%	259200
Ran 500 7	500	50	56	96	100	91	62.5%	259200
Ran 500 8	500	50	56	96	100	91	62.5%	259200
Ran 500 9	500	50	56	96	100	91	62.5%	259200
Ran 500 10	500	50	56	96	100	91	62.5%	259200
Ran 500 11	500	150	5	143	200	104	1980%	259200
Ran 500 12	500	150	5	143	200	103	1960%	259200
Ran 500 13	500	150	5	143	200	104	1980%	259200
Ran 500 14	500	150	5	143	200	103	1960%	259200
Ran 500 15	500	150	5	143	200	103	1960%	259200
Ran 500 16	500	150	5	143	200	103	1960%	259200
Ran 500 17	500	150	5	143	200	103	1960%	259200
Ran 500 18	500	150	5	143	200	103	1960%	259200
Ran 500 19	500	150	5	144	200	103	1960%	259200
Ran 500 20	500	150	5	143	200	103	1960%	259200

3.5 Conclusions and Future Work

In this work we applied an elaborate `VNS` framework for the first time over 80 of the largest data instances of the MaxMinMin p-dispersion problem. We first explained a detailed preliminary experimental setting which captured a vast number of possibilities within the `VNS` framework, and then selected empirically the most promising setting in order to conduct the final experiments. We also incorporated a new plateau search module within the `VNS` framework coupled with elaborate shake functions. Unlike the previous methods, our computational experiments on the largest benchmarks found *all* of the best known solutions with the proof of optimality for 53 out of 80 instances for the first time in the literature.

Of course in order to have a more precise comparison of various heuristics the same executable codes of the different authors should be run under the same conditions with the same stopping criteria. Yet, we believe that the reported results represent a fair comparison with other state-of-the-art heuristics, as we have considered the best ever solutions obtained by all the heuristic methods ever studied in the literature regardless of their longer running times or the use of other internal heuristics. Besides our results clearly demonstrate the high performance of our solutions both in terms of quality and computational times. Owing to the efficient use of information (α and β values) the complexity of our `LocalSearch` method is very low (rarely quadratic in practice), besides a very efficient `Update` procedure with linear complexity that is even implemented in constant time for most cases. Another important point is that in the previous methods lower running times usually led to a decrease in the number of best solutions found, whereas in our work fast results go hand in hand with high quality solutions.

One of the most interesting advantages of the Variable Neighborhood Search metaheuristic is its flexibility and how it allows the decision maker to define and adapt the framework to its own problem specifications. For instance, in case of limited computational resources one might be interested in doing single runs but allowing longer running times to secure the chances of finding the best solutions. The choice of the best setting is always a matter of time and available computational resources and one could further investigate finer tuning of the parameters embedded in the proposed framework. As the future work we suggest developing

decomposed VNS frameworks in order to tackle large problem instances in a shorter time and to possibly further improve or even prove the optimality of the few remaining instances.

Acknowledgements

This research was funded by NSERC (Natural Sciences and Engineering Research Council of Canada) grant PGSD2-392404-2010, and FQRNT (Fonds de recherche du Québec - Nature et technologies) grant 134582. Pierre Hansen has been partially supported by NSERC grant 105574-2007, Sylvain Perron has been partially supported by NSERC grant 327435-06, and they were both partially supported by FQRNT team grant PR-131365.

Chapter 4

Bi-Objective Variable Neighborhood Search for the p-Diversity-Proximity Problem

Behnaz Saboonchi

Pierre Hansen

Sylvain Perron

Department of Management Sciences

GERAD and HEC Montréal

3000, chemin de la Côte-Sainte-Catherine

Montréal, Québec, H3T 2A7, Canada

Abstract

Application of the dispersion models in order to address the cannibalization phenomenon within franchised chains is a recent approach. These models focus on the maximization of dispersion among the same-brand units without considering their proximity to the customer zones. In this work we have designed a bi-objective location model which is aimed at maximizing the minimum distance among the newly located units while minimizing their gravity-based distance to the customer zones. This unique model captures simultaneously two of the most important concerns within the franchise location domain, for the first time in the literature. We finally develop a heuristic solution procedure based on the Bi-objective Variable Neighborhood Search (`BOVNS`) framework in order to create high-quality solutions for both objectives in the Pareto front. Extensive computational experiments are also presented.

4.1 Introduction

Dispersion models maximize dispersion as a function of the distance/dissimilarity among the entities in a network and are traditionally used to locate undesirable and noxious or obnoxious facilities [32, 34]. Erkut and Neuman [33] propose four different types of dispersion models based on different dispersion metrics. The first one is the MaxMinMin problem which maximizes the minimum distance between each pair of the selected facilities. The second one is the MaxSumMin which seeks to maximize the sum of the minimum distances from each facility to its closest neighbor. The third formulation is called MaxMinSum which takes the sum of the distances from each facility to all its neighbors and maximizes the smallest sum of the distances. Finally the fourth formulation corresponds to the MaxSumSum which aims at maximizing the sum of all the hub distances for all the located facilities.

Saboonchi et al. [76, 77, 78] apply the MaxMinMin, MaxMinSum and MaxSumSum dispersion problems for the first time to address the franchise cannibalization issue. They developed several heuristics based on the Variable Neighborhood Search metaheuristic framework including various greedy constructive procedures and different shaking strategies. They discuss the tradeoffs among different solution strategies and the comparison of results with those of the state-of-the-art heuristics demonstrate the high performance of their approaches.

To the best of our knowledge, the concept of the gravitational attraction functions for the candidate locations has never been incorporated into dispersion problems. The general dispersion concept is suitable for locating obnoxious and undesirable facilities, but not necessarily for locating retail stores. This is because one cannot focus only on the dispersion of the facilities without considering the relative distance to the customer zones. Therefore, the franchise location models need to be adapted to embrace both important factors involved in the retail networks: dispersion of the facilities and accessibility/utility perceived by the customers.

In classical location-allocation models it is normally assumed that the clients always prefer to choose the closest facilities in order to receive their desired services. This assumption is appropriate for locating fire stations or hospitals for instance, but not for locating retail or commercial centers, since clients behave according to a gravity-based formula to choose among different competing facilities [12].

In 1933 Walter Christaller [14], a German geographer, developed the Central Place Theory in order to explain the distribution patterns, size and the number of towns and cities around the world. The theory consists of two basic concepts: 1) the threshold of the demand in order to make the market profitable, and 2) the average maximum distance that the customer is willing to travel. Another location theory which has been widely used in order to locate commercial centers is the gravity model. The law of retail gravitation of Reilly [17, 73] has been derived from Newton's law and has been used to explain various types of behavior that occur between different entities or locations. A probabilistic view of the deterministic law of gravitation led to the Huff model [50, 51] and later to a generalized version called the Multiplicative Competition Interaction (MCI) model [66]. The MCI probabilistic version is based on the Huff formulation and also the market share models. For more comprehensive explanations please see [40].

Based on the gravity rule, instead of assigning customers to their closest facility, they are allocated to the points in inverse proportion to some function of the distance or the traveling time [53]. In other words each facility has a utility for each customer. The nature of the utility varies based on the type of the facility such as the distance/traveling time, physical store environment, closeness to other type of stores, variety, etc. [5, 10, 55, 56, 60].

Besides, the utility might depend on the socioeconomic and demographic factors such as the per capita income, age, gender, household size, the local unemployment rate, consumer mobility and population density, etc. [23, 41, 49, 52, 69].

The gravity-based formulation has been used in the p-median and equity problems [24, 25, 26], competitive location models [8, 9, 23, 28, 29, 30, 42] and airline p-hub location models [22, 31]. In this work we capture the cannibalization issue based on the MaxMinMin p-dispersion criterion as discussed in [76], and the proximity to the customer zones is modeled with a gravity-based p-median problem as suggested in [24]. As the result, throughout the paper the problem is called the *p-diversity-proximity model*. It should be noted that we consider the classical formulation of the MaxMinMin p-dispersion model; however, other dispersion metrics could also be considered based on the preferences of the decision maker.

In Section 4.2 we describe the problem and its mixed integer non-linear formulation. Section 4.3 presents a detailed explanation of the proposed `VNS` heuristic solution procedure for the p-diversity-proximity problem and then the computational experiments on the classical datasets are discussed. Finally we conclude the paper by highlighting our contributions and then propose suggestions for future research.

4.2 Problem Statement and Mathematical Formulation

This section presents the mathematical formulation for the p-diversity-proximity problem. Let $V = \{v_i, \forall i = 1, \ldots, n\}$, be a set of n vertices (potential locations/customer zones) and v_i representing each member of this set. It is assumed that the facilities and the (center of) customer zones can only be located on the network nodes. We define S as any subset of p vertices such that $S \subseteq V, |S| = p$. The subset of the vertices *not* present in the current solution is defined as $\bar{S}$ such that $\bar{S} = V \setminus S, |\bar{S}| = n - p = m$. For easier presentation purposes throughout the paper the size of the vertices outside the solution set is presented by m.

The clients could be present at the same nodes as the facilities. As the result we define $d(v_i, v_j)$ as the distance between points i and j, i.e. between any pair of the located facilities, or between the customer zone at node i and the facility located at node j. The utility that each customer i receives from each facility j is presented by $U(v_i, v_j)$ that could depend on

any of the factors discussed in Section 4.1. All the distances are measured based on the center of the area representing the facility or the demand point [24], and the clients' demand at each point is presented by w_i.

The first objective function $f_1(S)$ within the bi-objective model is defined as the smallest distance among all the p selected vertices (facilities) in S:

$$f_1(S) = \min_{v_i, v_j \in S} \{d(v_i, v_j)\}.$$

The portion of the demand at customer zone i that is served by the facility k is:

$$w_i \frac{U(v_i, v_k)}{\sum_{j=1}^{n} U(v_i, v_j) x_j},$$

where x_j is a binary decision variable defining if vertex v_j (facility at location j) is selected. Therefore, the second objective function $f_2(S)$ representing the total utility-based distance traveled by all customers to the selected p facilities would be [24, 25]:

$$f_2(S) = \sum_{i=1}^{n} w_i \frac{\sum_{j=1}^{n} U(v_i, v_j) d(v_i, v_j) x_j}{\sum_{j=1}^{n} U(v_i, v_j) x_j}.$$

The bi-objective problem can be modeled as the following 0-1 mixed integer non-linear program as suggested in [25] and [32]:

$$\begin{aligned}
\max \quad & Z \\
\min \quad & \sum_{i=1}^{n} w_i \frac{\sum_{j=1}^{n} U(v_i, v_j) d(v_i, v_j) x_j}{\sum_{j=1}^{n} U(v_i, v_j) x_j} \\
s.t. \quad & Z \leq M(2 - x_i - x_j) + d(v_i, v_j) \quad 1 \leq i < j \leq n \\
& \sum_{j=1}^{n} x_j = p \\
& x_j = \{0, 1\} \qquad\qquad\qquad\qquad 1 \leq j \leq n.
\end{aligned}$$

In the above formulation M is a sufficiently large number which could be set as D_{max}, the largest distance among all the n vertices. An upper bounding technique in order to obtain tighter bounds is discussed in [76].

4.3 Bi-Objective Variable Neighborhood Search

Variable Neighborhood Search (VNS) is a metaheuristic or framework for building heuristics which is based on the idea of a systematic change of the neighborhood in order to escape from the valleys surrounding local optima, followed by a local search to find improved solutions. This general method has been proposed by Mladenović and Hansen [64] and has proven to lead to very successful heuristics for solving large combinatorial programs with applications in location theory, cluster analysis and several other fields. For a recent survey of the theoretical developments and applications including several hundred references see [45, 48].

The Variable Neighborhood Search method was first applied to the MaxMinMin p-dispersion problem (objective 1) in [76], coupled with new plateau search and shake modules. This method obtains high quality and robust solutions with low running times and computational complexity. As a result, we have used the same VNS modules in order to address the first objective within the bi-objective framework. The gravity p-median problem (objective 2) has been solved with the Steepest Descent and Tabu Search methods in [25], yet the VNS method has never been applied to this problem.

The Bi-objective Variable Neighborhood Search Method (BOVNS) represents a more general framework compared to the single-objective VNS. It includes two objective functions at a time, with different neighborhood structures and solution selection strategies [38, 57, 58]. This innovative framework is illustrated in Algorithm 4.1.

The solution is represented by an array of n indices corresponding to each vertex or candidate location, where the first p elements correspond to the subset of the current solution S. The solution space U is represented by the $\binom{n}{p}$ subsets of V with cardinality p. In order to apply VNS, a metric function is defined to evaluate the distance between any two solutions S and S':

$$\delta(S, S') = \delta(S', S) = |S \setminus S'|.$$

Based on the metric distance function defined above, the neighborhood of size k of a solution S is defined as:

$$N_k(S) = \{S' \in U | \delta(S, S') = k\}; k = 1, 2, \ldots, \min\{p, m\}.$$

Throughout this section the following notations are used:

- S_{cur}: the current solution set where $|S_{cur}| = p$;
- $f_1(S)$, $f_2(S)$: the first and second objective function values corresponding to the current solution set S;
- v_{exit}: the vertex inside the solution set that is a candidate to *leave* the solution set ($v_{exit} \in S$);
- v_{enter}: the vertex outside the solution set that is a candidate to *enter* the solution set ($v_{enter} \in \bar{S}$);
- $\alpha(v_i)$: the sum of the product of the utility and the distance from any customer zone ($v_i \in V$) to all the vertices in the solution set ($v_j \in S$):

$$\alpha(v_i) = \sum_{j=1}^{p} U(v_i, v_j) d(v_i, v_j), \forall i = 1, \ldots, n;$$

- $\beta(v_i)$: the sum of the utilities from any customer zone ($v_i \in V$) to all the vertices in the solution set ($v_j \in S$):

$$\beta(v_i) = \sum_{j=1}^{p} U(v_i, v_j), \forall i = 1, \ldots, n.$$

The `BOVNS` function is defined in Algorithm 4.1. The stopping criterion is the total running time. The already elapsed cumulative time in the overall procedure is noted by $t_{elapsed}$. The k_{min} and k_{step} (shake step size) parameters are set by default to 1, and the k_{max} (maximum shake size) is set to a coefficient of $\min\{p, m\}$, as discussed later.

The `BOVNS` framework starts by the `Initialize` function. The initial solution could be created at random or in a greedy manner. Based on the *Random add* method the initial

```
function BOVNS (k_min, k_step, k_max)
S_cur ← Initialize();
ParetoUpdate(S);
t_elapsed = 0;
while t_elapsed ≤ t_max do
    k_cur = k_min;
    S_cur ← Randomize();
    while k_cur ≤ k_max and t_elapsed ≤ t_max do
        S_cur ← Shake(S_cur);
        S_cur ← LocalSearch(S_cur);
        ParetoUpdate(S);
        if ParetoUpdate(S) then
            k_cur = k_min;
        else
            k_cur = k_cur + k_step;
        end
    end
end
```

Figure 4.1: Pseudo Code for the BOVNS Framework

solution is simply created by choosing p vertices at random. The *Greedy add* and the *Greedy drop* construction heuristics could have been applied as suggested in [76, 77, 78]. Yet, empirically the use of greedy initialization methods is not necessary as the elaborate local search function would soon find high quality solutions. The initial solution is the first member to be inserted in the approximate Pareto front as explained later in Section 4.3.1.

Then the `Randomize` function chooses a random neighborhood structure from the current approximate Pareto front before performing the `Shake` and `LocalSearch` procedures, i.e. a solution will be chosen randomly among the existing solutions in the current Pareto front. This helps diversify the solutions and better build the Pareto front. The `Shake` and `LocalSearch` procedures are designed differently for each of the objective functions. The `BOVNS` algorithm randomly chooses only *one* of the objectives at each iteration and performs the `Shake` and `LocalSearch` procedures accordingly. This method creates high quality solutions for both objectives while bringing in more diversity in the Pareto front.

The notion of the improvement in the bi-objective `VNS` is different from the single-objective one, since the improvement is made whenever the Pareto front is updated, i.e. a non-dominated solution enters the Pareto front. As the result, whenever the Pareto front is

updated, the step size k_{cur} is reset to k_{min}, otherwise it is incremented by k_{step}. The following subsections explain the `BOVNS` modules in details.

4.3.1 Pareto Front Update

In bi-objective optimization, unlike single-objective optimization, it is not possible to find a single global optimum while considering all the objectives [38]. Each solution generated within the `BOVNS` framework represents a value for the first objective (f_1), and another value for the second objective (f_2). The f_1 is a maximization objective, whereas f_2 is a minimization one. Therefore, two solutions S_1 and S_2 are called non-dominated solutions, i.e. are not superior to each other, if S_1 leads to a greater/better dispersion value for f_1 but greater/worse proximity value for f_2 compared to solution S_2, or vice vera. On the contrary, the solution S_1 dominates solution S_2 if it has better value in one of the objectives as well as equal or better value for the other objective.

In bi-objective optimization the goal is to find a set of non-dominated solutions or the approximate Pareto front (for formal explanations see [16, 61]). The `ParetoUpdate` function evaluates any solution S_{cur} and inserts it into the current Pareto front if it's not dominated. The `ParetoUpdate` function is called after the `Shake` function and during the `LocalSearch` procedures as explained in In Section 4.4.

4.3.2 Shake

The perturbation in most `VNS`-based heuristics is done in a simple manner by choosing a random vertex from the k_{th} neighborhood, i.e. $N_k(S)$ from the current solution S and then repeating k times the random swap move. The `RandomShake` function does so by choosing one random leaving and entering candidates at each iteration with updates in between each swap. However, we have developed two additional shake functions in order to control the perturbation operation in a more intelligent manner. The reader is referred to [76] for more details about the `Shake` functions for the MaxMinMin p-dispersion objective (f_1), yet the `Shake` functions for the gravity p-median objective (f_2) are explained in the following.

The `SemiGreedyShake` function fixes a *random* leaving candidate from the current solution set S and then chooses an entering candidate $v_i \in \bar{S}$ whose $\alpha(v_i)/\beta(v_i)$ value is the smallest. The α and β values are initially calculated per each customer zone at point i. Yet, we are also allowed to use these values to evaluate facility locations in terms of their attraction, since it is assumed that both facilities and customer zones could be located at each node i. This shake method does not guarantee an improvement in the objective function value, yet simply chooses a reasonable entering candidate after having fixed the leaving candidate. This procedure is repeated until the shake size of k is attained. Each iteration is performed in $O(m)$ time and after each swap the `Update` function is called.

In order to have a more intensified shake operation the `GreedyShake` function could be used which works the same way as the `SemiGreedyShake` function in order to find the entering candidates. Yet, the leaving candidate would be the one with the largest $\alpha(v_i)/\beta(v_i)$ value within the solution set. The performance of the shake functions in order to build the Pareto front is compared in Section 4.4 where it is demonstrated that the more intelligent shake functions dominate the purely random ones.

4.3.3 Local Search

The `LocalSearch` procedure for the MaxMinMin p-dispersion problem (f_1) is explained in [76]. The `LocalSearch` procedure for the gravity p-median objective (f_2) is implemented by performing 1-interchange swaps on the current solution as shown in Algorithm 4.2. This means that at each iteration only *one* vertex is swapped at a time. The swap could be done whenever the first (first improvement strategy) or the best (best improvement strategy) contribution is made to the current objective value. In this work the first improvement strategy is implemented in order to create more diversification. Besides, the first improvement strategy has much less iteration complexity in practice.

In order to start the `LocalSearch` procedure the gain obtained from swapping the selected entering candidate with the selected leaving candidate should be evaluated as explained in Section 4.3.3.1. The two main `Contribution` and `Update` functions will be explained in details in the following subsection.

```
function LocalSearch(S)
repeat
    (v_exit, v_enter, gain) ← Contribution(S);
    if gain > 0 then
        Swap(v_exit, v_enter);
        Update(v_exit, v_enter, S);
    end
until gain > 0;
```

Figure 4.2: Pseudo Code for the Local Search Procedure

4.3.3.1 Contribution and Update

In the proposed `LocalSearch` procedure the `Contribution` function can determine the first or best entering candidate $v_{enter} \in \bar{S}$, as well as its corresponding contribution to the current objective function value. The `Contribution` function for the MaxMinMin p-dispersion problem (f_1) is explained in details in [76]. The `Contribution` function for the gravity p-median objective (f_2) starts by choosing a random leaving candidate $v_{exit} \in S$, and a random entering candidate $v_{enter} \in \bar{S}$. In order to efficiently calculate the objective function value after the swap, the following values should be stored and updated during the `BOVNS` algorithm:

$$\alpha(v_i) = \sum_{j=1}^{p} U(v_i, v_j) d(v_i, v_j), \forall i = 1, \ldots, n,$$

$$\beta(v_i) = \sum_{j=1}^{p} U(v_i, v_j), \forall i = 1, \ldots, n.$$

The `Update` procedure performs all the required updates before proceeding to the subsequent iteration. It is implemented in two different phases in order to update all the $\alpha(v_i)$ and $\beta(v_i)$ values, and then to update the objective function value $f_2(S_{cur})$. The update for the $\alpha(v_i)$ and $\beta(v_i)$ after each swap of $v_{exit} \in S$ and $v_{enter} \in \bar{S}$ is straightforward and is performed in $O(n)$ total time [25]:

$$\alpha(v_i) = \alpha(v_i) + U(v_i, v_{enter}) d(v_i, v_{enter}) - U(v_i, v_{exit}) d(v_i, v_{exit}), \forall i = 1, 2, \ldots, n,$$

$$\beta(v_i) = \beta(v_i) + U(v_i, v_{enter}) - U(v_i, v_{exit}), \forall i = 1, 2, \ldots, n.$$

The second objective function (f_2) is calculated in $O(np)$ time only for the first time, and then its updated value is calculated in $O(n)$ time at each iteration:

$$f_2'(S_{cur}) = \sum_{i=1}^{n} w_i \frac{\alpha(v_i)}{\beta(v_i)}.$$

The gain in the `LocalSearch` procedure is the difference between $f_2(S_{cur})$ and $f_2'(S_{cur})$ (before and after each swap), and considering the minimization objective, an improvement is made whenever the gain is positive, i.e. $f_2(S_{cur}) > f_2'(S_{cur})$.

4.4 Computational Experiments

In this section the proposed `BOVNS` is applied on 40 classical p-median test problems (pmed1 to pmed40) by Beasley [6, 7]. The total number of potential locations varies from n=100 to 900, and the value of p from 5 to 200. The utility function is a decay function of the distance which for comparison purposes is equal to $U(d) = \frac{1}{d^2+1}$, and the w_i is set to 1 as suggested in [25]. The value of "1" is added in the denominator in order to avoid "divisions by zero".

First we describe our experiments that were designed to study the performance of different settings within the `BOVNS` framework and then analyze the overall results obtained by different methods over all the test problems. The proposed `VNS` for the MaxMinMin p-dispersion problem (f_1) has already been discussed in details in [76], yet the suggested `VNS` for the gravity p-median problem (f_2) has never been addressed before. Therefore, we discuss the results for the gravity p-median problem separately in Section 4.4.2.

4.4.1 Results and Analysis

The `BOVNS` algorithm starts with the Random add method. There are three different shaking possibilities for both objectives: Random shake, Semi-Greedy shake and Greedy shake, and the `LocalSearch` procedure pursues a first improvement strategy favoring more diversification. Finally, the maximum shake size is set to: $k_{max} = 0.5 * \min\{p, m\}$.

In order to better study the size of the Pareto front and the quality of the solutions for both objective functions, we have designed two sets of experiments: 1) `BOVNS` including the above-explained three shake methods and the `LocalSearch` function leading to the *(Random-LS)*, *(SemiGreedy-LS)* and *(Greedy-LS)* methods and 2) `BOVNS` including only the three shake functions without any local search procedure leading to *(Random-RS)*, *(SemiGreedy-RS)* and *(Greedy-RS)* methods. The second set of experiments follows the *Reduced VNS* method [45], favoring only the advancement of the Pareto front without necessarily putting any specific effort in the `LocalSearch` procedure.

The heuristics are coded in C++ and compiled with the $-O2$ optimization option and are run on a linux machine with 2.667 GHz and 3Gb Ram. Each method is run 5 times for each of the 40 instances and the stopping criteria is the total running time which is set to 5 hours for the pmed1 to pmed20 instances, and to 10 hours for the pmed21 to pmed40 instances.

Table 4.I summarizes the results obtained by each of the six methods. The first column specifies the name of the method and the second and third columns represent the best (largest) size of the Pareto front and the average size of the Pareto front in five runs, averaged over all the 40 instances. The fourth and fifth columns show the best (largest) and average values for the MaxMinMin p-dispersion objective (f_1) respectively. The last two columns represent the best (smallest) and average values for the gravity p-median objective (f_2) respectively.

As it is clear from the values marked in bold in Table 4.I , the first three `BOVNS` methods *including* the `LocalSearch` procedure lead to better *average* results in terms of the size of

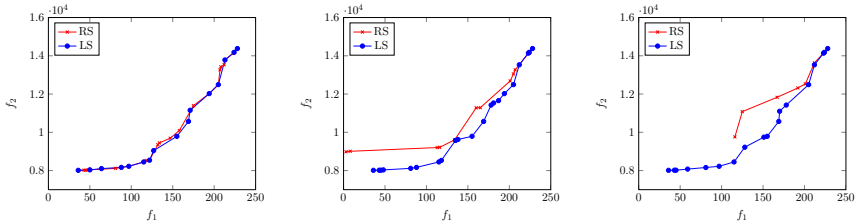
Table 4.I : Comparison of the BOVNS Methods

Method	Pareto-Best	Patero-Ave	f_1-Best	f_1-Ave	f_2-Best	f_2-Ave
Random-LS	13.03	12.17	60.7	60.47	10118.04	10118.04
SemiGreedy-LS	14.6	12.65	66.33	66.12	10118.04	10118.04
Greedy-LS	16.25	14.67	66.55	64.67	10118.04	10118.04
Average	**14.63**	**13.16**	**64.53**	**63.75**	**10118.04**	**10118.04**
Random-RS	11.83	10.65	53.95	52.71	10133.79	10138.07
SemiGreedy-RS	8.8	7.72	66.5	66.37	11451.78	11634.49
Greedy-RS	11.38	10.21	66.63	66.6	12488.29	12819.29
Average	10.67	9.53	62.36	61.89	11357.95	11530.61

the Pareto front and the quality of the solutions for both objectives (best known solutions are shown in Table 4.II). It should be noted that the *(Greedy-RS)* method finds slightly better results for some of the larger instances only for the first objective (f_1) compared to the *(Greedy-LS)* method. Yet, the focus here is not on a slight superiority of a specific method for an individual objective function and the goal is to consider both objectives and the size of the Pareto front.

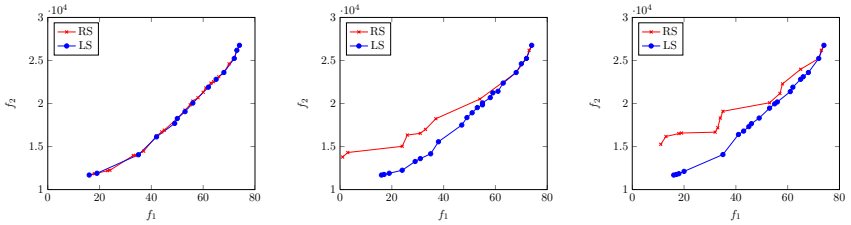
In order to better visualize the Pareto front for different instance sizes, Figures 4.3 to 4.5 illustrate the overall Pareto front obtained from the 5 runs for pmed1, pmed21 and pmed40 instances with small, medium and large sizes respectively. Each figure constitutes of three sub-figures for each of the three shaking methods, while including two plots for each to compare the two `BOVNS` strategies *with* the `LocalSearch` procedure (LS) and *without* the `LocalSearch` procedure (RS). All the sub-figures have an increasing slope which illustrates the non-dominance among the solutions in the Pareto front.

Based on the figures, the (LS) methods dominate or equal the (RS) methods for all the three instances. The dominance in terms of the size of the Pareto front and the quality of the obtained solutions becomes more evident as the size of the instances grow. Another observation is that in order to find better solutions for larger instances, more elaborate shake functions should be used (Random shake function alone is not capable of finding the best known solutions). Overall, the most promising method in terms of the quality of the solutions and the size of the Pareto front, is the *(Greedy-LS)* which includes the `LocalSearch` procedure and performs a Greedy shake for both objectives. It should be noted that the best known solutions for both objectives as shown in Table 4.II are: $f_1 = 228$ and $f_2 = 8012.115$ (for pmed1), $f_1 = 74$ and $f_2 = 11691.122$ (for pmed21) and $f_1 = 23$ and $f_2 = 14359.427$



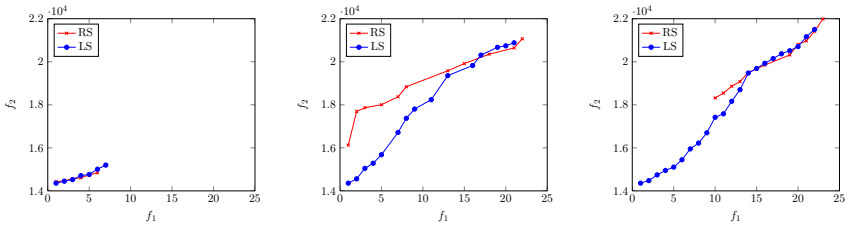
(a) Random Shake (b) SemiGreedy Shake (c) Greedy Shake

Figure 4.3: Comparison of the Six Methods for pmed1



(a) Random Shake (b) SemiGreedy Shake (c) Greedy Shake

Figure 4.4: Comparison of the Six Methods for pmed21



(a) Random Shake (b) SemiGreedy Shake (c) Greedy Shake

Figure 4.5: Comparison of the Six Methods for pmed40

(for pmed40), and our bi-objective methods obtain the best known results for each of the individual objectives.

4.4.2 Gravity p-Median Problem

The proposed `VNS` for the gravity p-median problem has never been addressed before in the literature and the studied datasets have never been used for the MaxMinMin p-dispersion problem. Therefore, the results for both objectives are discussed in more details here.

In order to test the single-objective `VNS` for the gravity p-median problem (f_2), the algorithm explained in Section 4.2 is run using the three Random shake, Greedy shake and SemiGreedy shake functions, as well as three different maximum shake sizes of $k_{max} = 0.5 * \min\{p, m\}, 0.75 * \min\{p, m\}$ and $min\{p, m\}$. As the result, each instance is run under the above nine settings only *once* with a total running time of $5 \times n$ seconds. The same setting has also been applied for the MaxMinMin p-dispersion problem (f_1).

In the first three columns of Table 4.II , the name and specifications of the instances are shown. The fourth, fifth and sixth columns represent the best solutions found in a *single* run of the above-explained nine settings, the average results obtained by all the nine

Table 4.II : Individual Objective Function Best Known Values

Instance	n	p	Best(f_2)	Ave.(f_2)	Time(f_2)	Best(f_1)	Ave.(f_1)	Time(f_1)
pmed1	100	5	8012.115	8012.115	0.11	228	226.667	0
pmed2	100	10	6850.371	6850.371	0	181	181	1
pmed3	100	10	7071.6	7071.6	0	167	165	0
pmed4	100	20	6437.958	6437.958	0	125	125	0
pmed5	100	33	3447.913	3447.913	0	75	75	0
pmed6	200	5	10120.714	10120.714	0	159	159	0
pmed7	200	10	8902.829	8902.829	0.11	118	118	0
pmed8	200	20	8699.655	8699.655	0	92	92	0.22
pmed9	200	40	6778.697	6778.697	0.11	62	61.778	76.78
pmed10	200	67	4268.302	4268.302	0.44	33	32.556	62.44
pmed11	300	5	9834.83	9834.83	0.44	112	112	0
pmed12	300	10	10279.79	10279.79	0.56	92	92	0
pmed13	300	30	9059.974	9059.974	0.67	64	63.778	81.56
pmed14	300	60	8301.225	8301.225	4.44	43	42.333	161
pmed15	300	100	6271.2	6271.252	2.33	27	26.556	215.11
pmed16	400	5	10404.405	10404.405	0.56	91	91	1
pmed17	400	10	10516.011	10516.011	0.67	71	71	0
pmed18	400	40	10644.121	10644.471	1.33	48	47.667	162.56
pmed19	400	80	8794.938	8794.938	2.22	31	30.333	293.56
pmed20	400	133	7531.657	7531.681	5.78	21	20.333	1
pmed21	500	5	11691.122	11691.122	0.22	74	74	0
pmed22	500	10	12640.371	12640.371	0.33	66	65.667	1
pmed23	500	50	11008.56	11008.56	3.67	39	38.222	390.67
pmed24	500	100	9780.748	9780.786	6.56	25	24.333	270.33
pmed25	500	167	8291.135	8291.135	59.44	17	16.222	75.67
pmed26	600	5	12444.871	12444.871	0.44	68	68	0
pmed27	600	10	11973.176	11973.176	0.44	59	59	0
pmed28	600	60	11057.149	11057.294	5.89	31	30.667	384.67
pmed29	600	120	10482.444	10482.485	12.67	22	21.333	87
pmed30	600	200	9975.692	9975.75	98.22	15	14.333	1.33
pmed31	700	5	12909.707	12909.707	0.56	57	56.667	1
pmed32	700	10	13550.981	13550.981	0.89	52	51.333	0.44
pmed33	700	70	12347.303	12347.321	18	27	26.667	122
pmed34	700	140	11206.771	11206.771	23	19	18.111	108.89
pmed35	800	5	13060.535	13060.535	0.56	58	58	0.22
pmed36	800	10	14360.506	14360.506	2.33	51	51	0
pmed37	800	80	13810.45	13810.45	17.67	27	26.222	313
pmed38	900	5	13990.942	13990.942	1.44	57	57	1
pmed39	900	10	13551.33	13551.33	1.56	41	41	1.11
pmed40	900	90	14359.427	14359.611	22.44	23	22.333	1
Average			10118.04	10118.06	7.4	66.7	66.33	70.39

settings and the average time (over the nine settings) when the best solutions were found for the gravity p-median objective, respectively. Here the best obtained results are exactly the same as the best known values reported in [25]. The last three columns demonstrate the same results for the MaxMinMin p-dispersion objective. All the various nine settings almost always obtain the best known solutions for both objectives (10118.04 vs. 10118.06 and 66.7 vs. 66.33). Besides, the best known values for the gravity p-median problem are found in a very short average computational time (the exact comparison with the literature is not possible as the other methods reported in [25] have structural differences with our methods and have different stopping criteria, etc.). It is worthwhile to mention that in this work the single objective values are not of our concern and thus, the overall high performance of the bi-objective model is taken into account.

4.5 Conclusions and Future Work

In this work we presented an innovative model within the franchise location domain, which addresses the cannibalization issue and the proximity to the clients under a bi-objective optimization model. Several heuristics have been discussed in order to create high quality solutions within the Pareto front using the Bi-objective Variable Neighborhood Search method (`BOVNS`). The proposed `BOVNS` is a new way to address two objectives in the `VNS` framework and could easily be expanded to solve multiobjective optimization problems.

The dispersion among the facilities is captured using the MaxMinMin (p-dispersion) model and the proximity to the clients is modeled by the gravity p-median problem. The second objective function is non-linear and the use of heuristic methods helps find high quality solutions even for larger instances. To the best of our knowledge, this work is the first application of the `VNS` to the gravity p-median problem and our methods find all the best known results in the literature in a very short time. The inclusion of the `LocalSearch` procedure and the Greedy shakes embedded in the `BOVNS` framework help find dispersed solutions within the Pareto front while reaching most of the best known solutions for both objectives.

This work is a step towards more applied location models within the franchise location domain and the decision makers could simply feed their real life data in terms of attraction

functions, distances, locations, etc. into the proposed model. As future research we propose more elaborate `BOVNS` modules and finer tuning of its parameters in order to create even more dispersed solutions within the Pareto front. The proposed model is very flexible and can easily be adapted to include other dispersion metrics. Therefore, another interesting future work would be the application of the proposed model in real life and practical problems and to perhaps consider other dispersion metrics.

Acknowledgements

This research was funded by NSERC (Natural Sciences and Engineering Research Council of Canada) grant PGSD2-392404-2010, and FQRNT (Fonds de recherche du Québec - Nature et technologies) grant 134582. Pierre Hansen has been partially supported by NSERC grant 105574-2007, Sylvain Perron has been partially supported by NSERC grant 327435-06, and they were both partially supported by FQRNT team grant PR-131365.

General Conclusion

Considering the importance of the cannibalization effects within the franchise location domain, this thesis proposes a new perspective to tackle this issue. The suggested method is the utilization of the dispersion concepts which are designed to create dispersed location solutions. This constitutes a novel perspective of the classical dispersion models in this context.

Based on the extensive literature review, it is known that the dispersion problems impose difficulties on exact and heuristic solution procedures especially for larger problem sizes. The proposed methods in this work are shown to be very effective to solve large instances and our methods compare favorably with the state-of-the-art heuristics in terms of the quality and robustness of the obtained solutions, lower running times and computational complexity.

The heuristic algorithm developed for the p-dispersion-sum (MaxMinSum) is the first solution procedure addressing this problem, setting new benchmarks for future research. The computational experiments on the small to large-sized instances mainly study the tradeoffs between the diversification and intensification `VNS` modules. It is concluded that the greedy components within the `VNS` framework would lead to better results in longer running times. On the contrary, the random modules would increase the chances of obtaining more diversified and improved solutions in repeated runs. The choice of the best setting is based upon the discretion of the decision maker and the available computational resources.

The maximum diversity problem (MaxSumSum) has been well studied by various state-of-the-art heuristics, and thus imposing a challenge for new heuristics in order to further improve the benchmark results. Our results reveal that the plateau search mechanism added to the basic `VNS` framework plays an important role to improve the quality of the solutions.

Besides, the use of more intelligent shake functions is another successful technique leading to several new benchmark results in the literature.

The p-dispersion problem (MaxMinMin) is another classical model that has been addressed by several exact and heuristic solution procedures. We developed the first `VNS` approach with low computational complexity and an efficient update function. Our method finds all the best known benchmark solutions with the proof of optimality for most of the instances. Like the previous models, the results confirm the necessity of the plateau search mechanism and the greedy-based shake functions.

The bi-objective p-diversity-proximity problem is an innovative approach towards more practical location optimization models. The proposed model is a very flexible decision support tool allowing the decision makers to use real data and to choose among various tradeoffs and scenarios. Besides, our proposed bi-objective Variable Neighborhood Search framework is a new way to address multiple objectives in the `VNS` framework. The results indicate that as the size of the instances grow, the use of greedy-based shake functions and the utilization of the local search procedure (as opposed to the reduced `VNS` method that does not include the local search function) become more and more important in terms of the size and the quality of the non-dominated solutions within the Pareto front.

In order to illustrate a general overview of all the above suggested `VNS` methods for the MaxMinSum (Chapter 1), MaxSumSum (Chapter 2) and MaxMinMin (Chapters 3 and 4) problems, we have designed a comparison test based on the large datasets studied in the previous chapters. The `VNS` configuration used in the final tests follow the first improvement strategy for the local search procedure, and starts from the current solution at the beginning of each iteration. For each problem we compared the three different shaking methods over three different running times of 2, 5 and 10 hours, leading to nine various tests. The maximum shake size is selected based on each problem's characteristics. The objective here is to highlight the methods that find the best known results in the above running times. Since we do a single run of each setting, any method that finds results within 0.5% of the best known solutions is selected. Besides, ILOG CPLEX is run for the same 2, 5 and 10 hours of running time to provide a comparison basis with exact methods.

Figure C1 shows the test results for the MaxMinSum (Chapter 1) problem. The largest dataset studied in this chapter is the MDG-c containing 20 instances with $n = 3000$ and $p = 300, 400, 500$ and 600. Since several instances have the same n and p values, we have selected the p/n ratio as the Y axis, and the instance number as the X axis for illustration purposes. It should be noted that the instance numbers are presented in the same order that they appear in the dataset. The results reveal that:

- The longer running times do not make a difference within the 0.5% of the best known solutions.
- The SemiGreedy Shake method finds all the best known results (within 0.5%).
- The Random Shake and Greedy Shake methods find the best known results (within 0.5%) only for instances 14 to 20. One possible reason could be the fact that the Random Shake method is not strong enough to do intelligent moves, and the Greedy Shake method is too intensified for this specific problem and gets caught due to lack of diversification.
- Results obtained by ILOG CPLEX are never within the 0.5% of the best known solutions.

Figure C2 presents the same results for the MaxSumSum (Chapter 2) problem. MDG-c dataset is also one of the largest datasets used in this chapter with $n = 3000$ and $p = 300, 400, 500$ and 600. Same illustration method has been used in order to present the X axis and the Y axis. Besides, all the tests include the plateau search mechanism explained in Chapter 2. The results reveal that:

- The longer running times do not make a difference within the 0.5% of the best known solutions.
- The SemiGreedy and Greedy Shake methods find all the best known results (within 0.5%).
- The Random Shake method finds the best known results (within 0.5%) only for instances 6 to 20. As explained before, this could be due to the fact that the pure Random Shakes are not capable of making intelligent swaps for the dispersion problems.
- Results obtained by ILOG CPLEX are never within the 0.5% of the best known solutions.

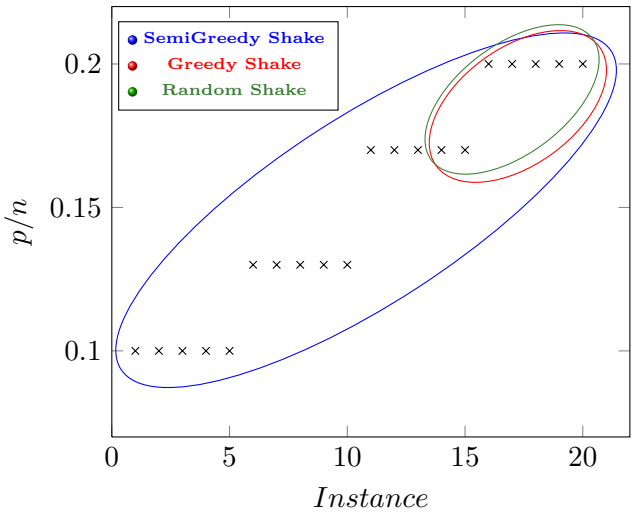


Figure C1: Comparison of the Three VNS Methods for MaxMinSum Problem

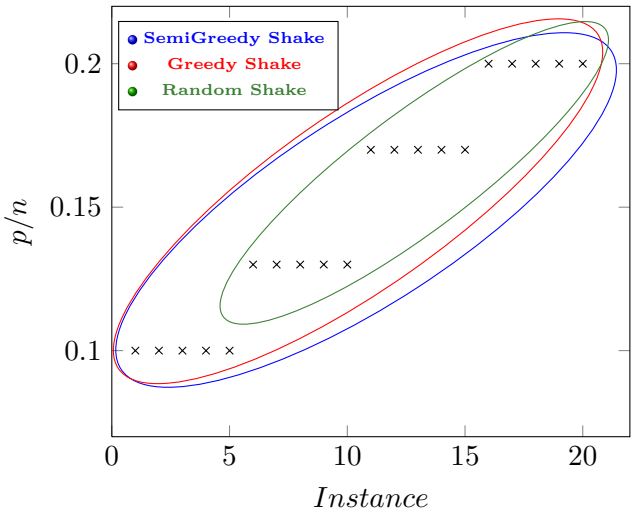


Figure C2: Comparison of the Three VNS Methods for MaxSumSum Problem

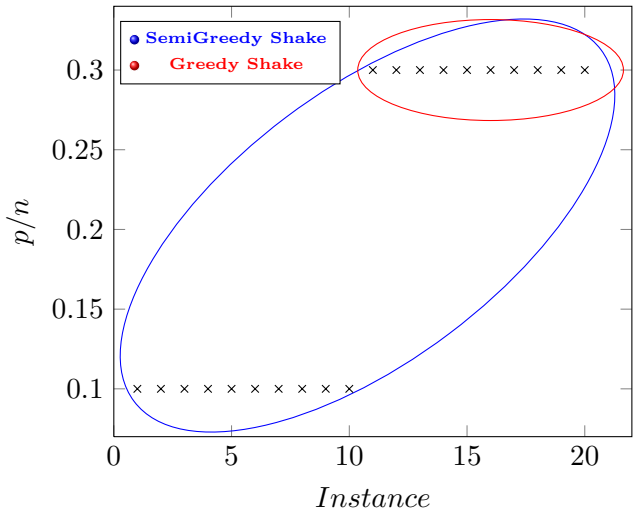


Figure C3: Comparison of the Three VNS Methods for MaxMinMin Problem

Finally, Figure C3 presents the results for the MaxMinMin (Chapters 3 and 4) problem. One of the largest and most challenging instances used in the chapter is the Ran dataset consisting of 20 instances with $n = 500$ and $p = 50, 150$. Same illustration method as the previous two models has been used in order to present the X axis and the Y axis. Besides, all the tests include the plateau search mechanism explained in Chapters 3. The results reveal that:

- The longer running times do not make a difference within the 0.5% of the best known solutions.
- The SemiGreedy Shake method find all the best known results (within 0.5%).
- The Greedy Shake method finds the best known results (within 0.5%) only for instances 11 to 20. Again this could be due to the intensified nature of this module which causes the algorithm to spend more computational time per iteration and thus to get caught in lower quality solutions.
- Results obtained by the Random Shake method and ILOG CPLEX are never within the 0.5% of the best known solutions. It should be noted that the distance values for this dataset are all integers, and thus finding results which are even one unit less than the best known solutions would not be within the 0.5% threshold.

The synthesis of the observations from Figures C1 to C3 would lead to the following integrated conclusions which have already been highlighted throughout the various chapters of the thesis:

- The shake function is an important component of the VNS framework and most of the VNS-based methods apply a random version of this function. Here, we pointed out that the utilization of more intelligent shake functions would significantly improve the quality of the obtained solutions. The SemiGreedy Shake function seems to perform better than the Greedy Shake method which could be due to the fact that it contains a combination of intensification and diversification modules.

- The data structure of the instances plays an important role in terms of the difficulty of the solution procedure. Although we have not found any consistent empirical rule between the p/n ratio and the number of obtained best known solutions, results demonstrate that smaller p/n ratios might be more challenging to solve.

- The exact methods are not capable of finding results within a reasonable percentage of the best known solutions which confirms the need for heuristic solution procedures to tackle large and challenging dispersion problem instances.

This thesis is another step towards more applied location models within the franchise location domain allowing flexible VNS modules, input data, parameters, etc., in order to embrace practical considerations. The elaborate local search and plateau search mechanisms, intelligent shake functions, low complexity update methods and the bi-objective framework created in this thesis would complement the existing modules within the Variable Neighborhood Search metaheuristic framework and can easily be used in order to solve various optimization problems. As future research we propose the application, modification and further elaboration of the proposed methods in order to solve other potential combinatorial problems, even including multiple objectives. Applying these methods to real industry problems would be another interesting path for future research.

Bibliography

[1] Achabal D, Gorr W, Maharajan V. Multiloc: A multiple store location decision model. Journal of Retailing 1982;58(2):5.

[2] Agcaand S, Eksioglu B, Ghosh J. Lagrangian solution of maximum dispersion problems. Naval Research Logistics (NRL) 2000;47(2):97–114.

[3] Aringhieri R, Bruglieri M, Cordone R. Optimal results and tight bounds for the maximum diversity problem. Foundations of Computing and Decision Sciences 2009;34(2):73–85.

[4] Aringhieri R, Cordone R. Comparing local search metaheuristics for the maximum diversity problem. The Journal of the Operational Research Society 2011;62(2):266–280.

[5] Bearden W, Netemeyer R, Teel J. Measurement of consumer susceptibility to interpersonal influence. Journal of Consumer Research 1989;15(4):473–481.

[6] Beasley J. A note on solving large p-median problems. European Journal of Operational Research 1985;21(2):270–273.

[7] Beasley J. Or-library: Distributing test problems by electronic mail. The Journal of the Operational Research Society 1990;41(11):1069–1072.

[8] Benati S. An improved branch & bound method for the uncapacitated competitive location problem. Annals of Operations Research 2003;122:43–58.

[9] Benati S, Hansen P. The maximum capture problem with random utilities: Problem formulation and algorithms. European Journal of Operational Research 2002;143(3):518–530.

[10] Bloch P, Ridgway N, Dawson S. The shopping mall as consumer habitat. Journal of Retailing 1994;70(1):23–42.

[11] Brimberg J, Mladenović N, Urošević D, Ngai E. Variable neighborhood search for the heaviest k-subgraph. Computers & Operations Research 2009;36(11):2885–2891.

[12] Carling K, Hakansson J. Short communication: A compelling argument for the gravity p-median model. European Journal of Operational Research 2012;In press.

[13] Chebat J, Chandon J. Predicting attitudes toward road safety from present and future time orientations: An economic approach. Journal of Economic Psychology 1986;7(4):477–499.

[14] Christaller W. Central places in southern Germany: Translated from Die zentralen Orte in Süddeutschland. Prentice Hall, 1966.

[15] Cliquet G. Implementing a subjective mci model: An application to the furniture market. European Journal of Operational Research 1995;84(2):279–291.

[16] Coello C. A comprehensive survey of evolutionary-based multiobjective optimization. Knowledge and Information Systems 1999;1(3):269–308.

[17] Converse P. New laws of retail gravitation. Journal of Marketing 1949;14(3):379–384.

[18] Curtin L, Church R. A family of location models for multiple-type discrete dispersion. Geographical Analysis 2006;38(3):248–270.

[19] Curtin L, Church R. Optimal dispersion and central places. Journal of Geographical Systems 2007;9(2):167–187.

[20] Della Croce F, Grosso A, Locatelli M. A heuristic approach for the max-min diversity problem based on max-clique. Computers & Operations Research 2009;36(8):2429–2433.

[21] Desarbo W, Kim J, Choi S, Spaulding M. A gravity-based multidimensional scaling model for deriving spatial structures underlying consumer preference/choice judgments. Journal of Consumer Research 2002;29(1):91–100.

[22] Drezner T, Drezner Z. A note on applying the gravity rule to the airline hub problem. Journal of Regional Science 2001;41(1):67–73.

[23] Drezner T, Drezner Z. Validating the gravity-based competitive location model using inferred attractiveness. Annals of Operations Research 2002;111:227–237.

[24] Drezner T, Drezner Z. Multiple facilities location in the plane using the gravity model. Geographical Analysis 2006;38(4):391–406.

[25] Drezner T, Drezner Z. The gravity p-median model. European Journal of Operational Research 2007;179(3):1239–1251.

[26] Drezner T, Drezner Z. The gravity multiple server location problem. Computers & Operations Research 2011;38(3):694–701.

[27] Duarte A, Martí R. Tabu search and grasp for the maximum diversity problem. European Journal of Operational Research 2007;178(1):71–84.

[28] Eiselt H, Gendreau M, Laporte G. Location of facilities on a network subject to a single-edge failure. Networks 1992;22(3):231–246.

[29] Eiselt H, Laporte G. Sequential location problems. European Journal of Operational Research 1997;96(2):217–231.

[30] Eiselt H, Laporte G, Thisse J. Competitive location models: A framework and bibliography. Transportation Science 1993;27(1):44–54.

[31] Eiselt H, Marianov V. A conditional p-hub location problem with attraction functions. Computers & Operations Research 2009;36(12):3128–3135.

[32] Erkut E. The discrete p-dispersion problem. European Journal of Operational Research 1990;46(1):48–60.

[33] Erkut E, Neuman S. Comparison of four models for dispersing facilities. INFOR 1991;29(2):68–86.

[34] Erkut E, Neuman S. A multiobjective model for locating undesirable facilities. Annals of Operations Research 1992;40:209–227.

[35] Erkut E, Ülküsal Y, Yeniçerioğlu O. A comparison of p-dispersion heuristics. Computers & Operations Research 1994;21(10):1103–1113.

[36] Fernández J, Pelegrín B, Plastria F, Tóth B. Planar location and design of a new facility with inner and outer competition: An interval lexicographical-like solution procedure. Networks and Spatial Economics 2007;7:19–44.

[37] Fernández J, Pelegrín B, Plastria F, Tóth B. Solving a Huff-like competitive location and design model for profit maximization in the plane. European Journal of Operational Research 2007;179(3):1274–1287.

[38] Geiger M. Randomised variable neighbourhood search for multi objective optimisation. In: Proceedings of the 4th EU/ME Workshop: Design and Evaluation of Advanced Hybrid Meta-Heuristics. 2004. p. 34–42.

[39] Ghosh A, Craig C. Fransys: A franchise distribution system location model. Journal of Retailing 1991;67(4):466–495.

[40] Ghosh A, McLafferty S. Location strategies for retail and service firms. Lexington Books Lexington, MA, 1987.

[41] González-Benito O, Greatorex M, Munoz-Gallego P. Assessment of potential retail segmentation variables an approach based on a subjective mci resource allocation model. Journal of Retailing and Consumer Services 2000;7(3):171–179.

[42] Hakimi S. Locations with spatial interactions: competitive locations and games. John Wiley & Sons, New York, 1990.

[43] Hansen P, Labbé M, Minoux M. The p-center-sum location problem. Cahiers du CERO 1994;36:203–219.

[44] Hansen P, Mladenović N. First vs. best improvement: An empirical study. Discrete Applied Mathematics 2006;154(5):802–817.

[45] Hansen P, Mladenović N, Moreno Pérez J. Variable neighbourhood search: methods and applications. Annals of Operations Research 2010;175:367–407.

[46] Hansen P, Mladenović N, Perez-Britos D. Variable neighborhood decomposition search. Journal of Heuristics 2001;7:335–350.

[47] Hansen P, Moon ID. Dispersing facilities on a network. Cahiers du GERAD 1995;.

[48] Hansen P, N. Mladenović J. Brimberg J, Pérez JM. Variable neighborhood search. In: Gendreau M, Potvin JY, editors. Handbook of Metaheuristics. Springer US; volume 146 of *International Series in Operations Research & Management Science*; 2010. p. 61–86.

[49] Hubbard R. A review of selected factors conditioning consumer travel behavior. Journal of Consumer Research 1978;5(1):1–21.

[50] Huff D. Defining and estimating a trading area. Journal of Marketing 1964;28(3):34–38.

[51] Huff D. A programmed solution for approximating an optimum retail location. Land Economics 1966;42(3):293–303.

[52] Ingene C, Yu E. Determinants of retail sales in smsas. Regional Science and Urban Economics 1981;11(4):529–547.

[53] Kang Y, Herr P, Page C. Time and distance: Asymmetries in consumer trip knowledge and judgments. Journal of Consumer Research 2003;30(3):420–429.

[54] Kaufmann P, Donthu N, Brooks C. An illustrative application of multi-unit franchise expansion in a local retail market. Journal of Marketing Channels 2007;14(4):85–106.

[55] Lafontaine F, Shaw K. Targeting managerial control: evidence from franchising. The Rand journal of economics 2005;36(1):131–150.

[56] Leszczyc PP, Sinha A, Sahgal A. The effect of multi-purpose shopping on pricing and location strategy for grocery stores. Journal of Retailing 2004;80(2):85–99.

[57] Liang Y, Chuang C. Variable neighborhood search for multi-objective resource allocation problems. Robotics and Computer-Integrated Manufacturing 2012;In press.

[58] Liang Y, Lo M. Multi-objective redundancy allocation optimization using a variable neighborhood search algorithm. Journal of Heuristics 2010;16(3):511–535.

[59] Lozano M, Molina D, García-Martínez C. Iterated greedy for the maximum diversity problem. European Journal of Operational Research 2011;214(1):31–38.

[60] Marjanen H. Longitudinal study on consumer spatial shopping behaviour with special reference to out-of-town shopping: Experiences from turku, finland. Journal of Retailing and Consumer Services 1995;2(3):163–174.

[61] Marler R, Arora J. Survey of multi-objective optimization methods for engineering. Structural and Multidisciplinary Optimization 2004;26:369–395.

[62] Martí R, Gallego M, Duarte A. A branch and bound algorithm for the maximum diversity problem. European Journal of Operational Research 2010;200(1):36–44.

[63] Martí R, Gallego M, Duarte A, Pardo E. Heuristics and metaheuristics for the maximum diversity problem. Journal of Heuristics 2011;doi: 10.1007/s10732-011-9172-4.

[64] Mladenović N, Hansen P. Variable neighborhood search. Computers & Operations Research 1997;24(11):1097–1100.

[65] Mladenović N, Labbé M, Hansen P. Solving the p-center problem with tabu search and variable neighborhood search. Networks 2003;42(1):48–64.

[66] Nakanishi M, Cooper L. Parameter estimation for a multiplicative competitive interaction model: Least squares approach. Journal of Marketing Research 1974;11(3):303–311.

[67] Palubeckis G. Iterated tabu search for the maximum diversity problem. Applied Mathematics and Computation 2007;189(1):371–383.

[68] Palubeckis G, Karčiauskas E, Riškus A. Comparative performance of three metaheuristic approaches for the maximally diverse grouping problem. Information Technology and Control 2011;40(4):277–285.

[69] Pan Y, Zinkhan G. Determinants of retail patronage: A meta-analytical perspective. Journal of Retailing 2006;82(3):229–243.

[70] Porumbel D, Hao J, Glover F. A simple and effective algorithm for the maxmin diversity problem. Annals of Operations Research 2011;186:275–293.

[71] Prokopyev O, Kong N, Martinez-Torres D. The equitable dispersion problem. European Journal of Operational Research 2009;197(1):59–67.

[72] Ravi S, Rosenkrantz D, Tayi G. Heuristic and special case algorithms for dispersion problems. Operations Research 1994;42(2):299–310.

[73] Reilly W. The law of retail gravitation. WJ Reilly, 1931.

[74] Resende M, Marttí R, Gallego M, Duarte A. Grasp and path relinking for the max-min diversity problem. Computers & Operations Research 2010;37(3):498–508.

[75] Ruiz J, Chebat J, Hansen P. Another trip to the mall: a segmentation study of customers based on their activities. Journal of Retailing and Consumer Services 2004;11(6):333–350.

[76] Saboonchi B, Hansen P, Perron S. Franchise location models and cannibalization effects: A variable neighborhood search approach. Les Cahiers du GERAD 2012;G-2012-60.

[77] Saboonchi B, Hansen P, Perron S. A greedy variable neighborhood search heuristic for the maxsumsum p-dispersion problem. Les Cahiers du GERAD 2012;G-2012-46.

[78] Saboonchi B, Hansen P, Perron S. Variable neighborhood search heuristics for the maxminsum (p-dispersion-sum) problem. Les Cahiers du GERAD 2012;G-2012-28.

[79] Silva G, Ochi L, Martins S. Experimental comparison of greedy randomized adaptive search procedures for the maximum diversity problem. In: Ribeiro C, Martins S, editors. Experimental and Efficient Algorithms. Springer Berlin / Heidelberg; volume 3059 of *Lecture Notes in Computer Science*; 2004. p. 498–512.

[80] Wang J, Zhou Y, Cai Y, Yin J. Learnable tabu search guided by estimation of distribution for maximum diversity problems. Soft Computing - A Fusion of Foundations, Methodologies and Applications 2012;16:711–728.