Les Cahiers du GERAD ISSN: 0711–2440

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

B. Saboonchi, P. Hansen,
S. Perron

G–2012–28

May 2012

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

Behnaz Saboonchi
Pierre Hansen
Sylvain Perron

GERAD & HEC Montréal
3000 chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada, H3T 2A7

behnaz.saboonchi@gerad.ca
pierre.hansen@gerad.ca
sylvain.perron@gerad.ca

May 2012

Les Cahiers du GERAD
G–2012–28

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 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 that 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.

Key Words: Combinatorial optimization, Dispersion problems, Metaheuristics, Variable Neighborhood Search.

Résumé

Les problèmes de dispersion consistent à choisir, parmi un ensemble de points, un sous-ensemble de taille donnée permettant de maximiser une fonction mesurant la distance entre les points choisis. De tels problèmes apportent un défi au niveau de la conception de méthodes de résolution heuristiques et métaheuristiques. Parmi les différentes variantes de problèmes de dispersion, les problèmes de MaxMin-Min (p-dispersion) et de MaxSumSum (*maximum diversity*) ont fait l'objet de plusieurs travaux de recherche alors que le problème de MaxMinSum a été très peu étudié. Dans cet article, nous nous attaquons au développement de méthodes heuristiques pour ce dernier problème. Ces méthodes sont basées sur la métaheuristique de recherche à voisinage variable (*Variable Neighborhood Search*) et diffèrent notamment au niveau des stratégies de recherche locale et de perturbation. Des tests empiriques nous permettent de comparer l'efficacité des différentes méthodes développées dans cet article. Enfin, une comparaison entre les résultats heuristiques et ceux d'une méthode exacte pour des instances de petite taille nous permet de confirmer la qualité des solutions heuristiques obtenues. À notre connaissance, il s'agit de la première application d'une méthode heuristique pour résoudre le problème de MaxMinSum. Ainsi, les résultats reportés dans cet article serviront de base de comparaison pour les développements futurs.

Acknowledgments: 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.

1 Introduction

In the family of the dispersion problems, given a set of n vertices we intend to select a subset of size p in a way that some measure of the distance 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 the missile silos where dispersion can reduce the chances of being all attacked by an attacker or for locating obnoxious facilities to be far from population zones [5]. The dispersion can also be a desirable factor when it comes to franchise location problems where we 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 [15].

Erkut and Neuman [6] 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 minimum 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. The 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 [8]. 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. [8] introduce two new variants of such problems and discuss their real life applications.

As mentioned by Curtin and Church [3], the MaxMinSum dispersion problem was first introduced in the location literature with the review of the dispersion objective metrics by Erkut and Neuman [6]. They [3] use this problem as part of their family of multiple-type dispersion formulations. 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 j.

This 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.

Due to this similarity, throughout this paper we would call the MaxMinSum problem as the "*p-dispersion-sum problem*". This problem was first modeled by Erkut and Neuman [6] 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 3.3 an upper bounding technique in order to obtain tighter bounds is discussed.

In Section 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 known benchmark test problems and also the comparison with exact methods in Section 3. Finally we conclude our paper by highlighting our contributions and suggestions for future research.

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 [13] 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 see e.g. [10].

Within the family of dispersion problems the VNS method has been applied only to the maximum diversity problem and has shown to be among the most efficient methods compared to other heuristics [2, 12]. 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 and fully connected graph $G(V, E)$, with $d_e \geq 0$ 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 apace 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. [2]. 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;
- S: the vertices present in the current solution set;
- $\bar{S}$: the vertices outside the current solution set;
- 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 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 noted 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 $\min\{p, n-p\}$.

```
function VNS (k_min, k_step, k_max, S);
S_best ← Initialize(S);
S_cur ← S_best;
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
```

Algorithm 1: Pseudo code for the VNS framework

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 at random p indices.

Two *Greedy* construction heuristics have been widely used in the literature in order to create initial solutions for the dispersion problems [7]. The *Greedy deletion* heuristic starts with all the n vertices and eliminates one vertex at each iteration. The objective function value improves by removing vertices, as the result for our problem the deletion candidate is the one with the smallest sum of the distances 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.

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 smallest decrease in the objective value [1, 2, 11]. 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 $\bar{C}$. At each iteration the objective function value is the smallest

$W(v_i)$ value for all the $v_i \in C$. 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 $v_i \in C$, and the newly added $W(v_{enter})$ value. By adding vertices to the under construction solution set the objective value will decrease or remain unchanged, as the result at each iteration the candidate vertex which will lead to the least decrease in the objective value is chosen.

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.

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 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);
gain = 0;
while gain > 0 do
    (v_exit, v_enter, gain) ← Contribution(S) ;
    if gain > 0 then
        Swap(v_exit, v_enter);
        Update(v_exit, v_enter, S);
    end
end
```

Algorithm 2: Pseudo code for the local search

2.2.1 Contribution

In Algorithm 3 the `Contribution` function in discussed. 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.

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 not 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

```
function Contribution(S);
gain ← 0;
choose a random v_exit and v_enter;
i = 0; j = p;
while i ≤ p do
    while j ≤ n − p do
        f(S_temp) = f(S_cur) ;
        if W(v_enter) − d(v_exit, v_enter) ≥ f(S_cur) then
            forall the (v_i ∈ S; i ≠ exit) do
                W(v_i) ← 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) ← min W(v_i);
            end
            if f(S_temp) > f(S_cur) then
                gain ← f(S_temp) − f(S_cur) ;
                return (v_exit, v_enter, gain);
                i = n;
                j = n − p;
            else
                j + +;
            end
        else
            j + +;
        end
    end
    i + +;
end
```

Algorithm 3: Pseudo code for determining the first improving swap and its contribution (first improvement strategy)

updated by: $W(v_{enter}) - d(v_{exit}, v_{enter})$ and will be added to the objective function, so if it's already smaller than the current objective function value (the already smallest sum of the distances) the new solution after the swap will be worse. As the result such a swap will be abandoned and the algorithm moves on to the next entering candidate.

If the swap candidate passes the preliminary check we are assured that the objective function value will remain unchanged or may improve after the swap. 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 the result in the worst case it is implemented in $O(p(n-p)p) = O(np^2)$ time per `LocalSearch` iteration. Of course the best improvement strategy will be implemented in exactly $O(np^2)$ time as every possible swap should be evaluated.

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.

2.3 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, 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, we need 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 calculate the overall objective function value for all the entering candidates and then select the insertion candidate that has the highest gain, or the lowest deterioration in the current objective function value. 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 $O((n-p)p)$. As the result this shake method does not guarantee that a deterioration in the objective function value would not occur, it simply chooses a reasonable entering candidate after having fixed the leaving candidate. Besides, with our method we add 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. In order to keep the random nature of the shake 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 3.

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. [12], 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. [16]. 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 [4] 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 [14].

First we describe our experiments that were designed to study the performance of different settings within the `VNS` framework and then we compare and analyze the tradeoffs and overall results obtained by different methods over all the test problems. Finally we compare our results with that of exact methods for relatively smaller instances.

All the heuristics were coded in C ++ and run on a linux machine, with 2.667 GHz and 24Gb Ram. The best obtained results after two hours of running time are reported as suggested in [12].

3.1 Experiments setup

As mentioned in Section 2 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 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 we are interested in trying a smaller value, i.e. $(0.75 * \min\{p, n-p\})$ to verify if it helps improve the performance of the heuristic. This will lead to $3 \times 3 \times 2 = 18$ combinations for 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 the 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 4. The average deviations presented in Tables 1 to 3 are all calculated based on the results presented in the fourth column (All methods) of Table 4 which refer to the best solutions obtained by the 72 combinations.

Table 1 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 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 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.

Table 1: 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

The above mentioned results for individual factors and settings do not necessarily lead to overall better combinations. As the result in Table 2 we present the same results as in Table 1 in a different manner by calculating the average deviations of the 18 combinations 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 [2, 9]. 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 solutions. As the result we prepared Table 3 which represents the number of best solution obtained over all the 80 data instances, for all the 72 different combinations.

From Tables 2 and 3 three important observations can be highlighted:

- The current iteration strategy and first contribution local search *(FC-FirstI)* leads to significantly higher number of best solutions obtained and the lowest average deviation compared to other `VNS` frameworks.
- The highest score among the settings belongs to the *(RA-SG-0.75p)* which refers to the Random start, Semi-Greedy shake, and 0.75p maximum shake size.
- The three best methods averaged over all the four `VNS` frameworks are *(GD-SG-0.75p)* , *(RA-SG-p)* and *(GD-SG-p)*.

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

Table 2: Average deviation for all combinations and frameworks

	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 3: Number of best solutions obtained for all combinations and frameworks

	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	

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 here 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 best 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. Surprisingly 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 is presented for each dataset in the fifth column (Multiple runs) of Table 4. Comparison of the fourth and the fifth columns of Table 4 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:

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

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 4: Best solutions for all the datasets

	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
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

Table 4: Best solutions for all the datasets (continued)

	n	p	All methods	Multiple runs	Long run
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
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 higher quality solutions. Here we take the same approach and run the preliminary experiments on the MDG-c largest dataset first. 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 we choose the *(GD-SG-p)* heuristic for the long run experiment 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 4 for each dataset. Comparison of the third and the fifth columns of Table 4 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 4, the best value obtained in the third column is higher than

the best solution in the fourth and the fifth columns. The reason is that the values reported in the third column corresponds to the best values under one run of all the 72 previously discussed combinations for two hours of running time, whereas the fourth column reports the best values after 30 runs of only the *(RA-SG-p)* heuristic for two hours of running time each, and the fifth column represents the values of one run of only the *(GD-SG-p)* heuristic for 10 hours of running time.

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 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 CPLEX with the best solution we ever obtained from our heuristics in Section 3 as an initial solution and also 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 its distance to its most distant $(p-1)$ vertices. Now if these values are sorted in increasing 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 CPLEX, we ran it only on the smallest SOM-b dataset and let it run as long as CPLEX is capable up to a maximum of two weeks running time. In the fourth column of Table 5 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 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, 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 5: Comparison with exact methods

	n	p	Lower bound	Upper bound	Best bound	Gap
SOM-b-1	100	10	62	81	81 (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%

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 section 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 we 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.

References

[1] 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.

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

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

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

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

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

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

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

[9] Hansen, P., Mladenović, N. First vs. best improvement: An empirical study. Discrete Applied Mathematics 2006;154(5):802–817. IV ALIO/EURO Workshop on Applied Combinatorial Optimization.

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

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

[12] Martí, R., Gallego, M., Duarte, A., Pardo, E. Heuristics and metaheuristics for the maximum diversity problem. Journal of Heuristics 2011;:1–2510.1007/s10732-011-9172-4.

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

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

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

[16] 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.