19.7/20 +4

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

# Selected Topics on Optimization

## Computational Experience with Heuristics for the PDP

# Report

Professor: Roger Zirahuen Rios Mercado Team E

| Members | ID | Degree |
|---|---|---|
| **Eros Vargas Mariscal** | 2226517 | ITS |
| **Angela Susana Torres Pasillas** | 2101844 | ITS |
| **Tadeo Alejandro Velázquez Vega** | 2226980 | ITS |
| **Cesar Maximiliano Peña Mendoza** | 2144135 | ITS |

Date: 15/05/2026

## Introduction

We consider the problem of selecting p out of n given candidate points, such that the minimum distance between pairs of the selected points is maximized. The objective is to disperse the selected points as much as possible.

This maximin problem is termed as “the p-dispersion problem" by Moon and Chaudhry (1984) in a survey paper on network location problems with distance constraints [1]. 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 [7].

Another application may be in the location of fast-food franchises in an urban area. The decision maker may have $n$ rental sites available and may want to open $p$ franchises. In this scenario, it may not be desirable to have two franchises located close to one another, sharing the same customer base. The total sales volume may be higher if the franchises are dispersed throughout the city [1].

Erkut and Neuman propose four different types of dispersion models based on different dispersion metrics:

1. The MaxMinMin problem which maximizes the minimum distance between each pair of the selected facilities.
2. The MaxSumMin which seeks to maximize the sum of the minimum distance from each facility to its closest neighbors.
3. The MaxMinSum which takes the sum of the distances from each facility to all its neighbors and maximizes the smallest sum of the distances.
4. The MaxSumSum which aims at maximizing the sum of all the hub distances for all the located facilities [7].

In the next section we will be describing the four different types Erkut and Neuman proposed, and the different heuristic solution procedures that have been discussed to solve the variation of dispersion problems. And following by the comparison Erkut [7] made between four categories of construction algorithms, he concluded that the neighborhood and interchange heuristic provide better results, in the case of dispersion models, than the construction heuristics.

## Problem Description

Following the standard framework for combinatorial optimization, the problem is defined by the following four elements:

1. Data (Input):

- In the p-dispersion problem, the data consists of a spatial configuration and a quantified measure of separation. We define the input parameters as follows:
  - The Candidate Set ($N$): A set of n predetermined potential locations or "nodes" in each space, represented as $N = \{1, 2, \ldots, n\}$.
  - The Population Parameter ($p$): A positive integer representing the exact number of facilities to be selected from a set $N$, where the condition $2 \leq p < n$ must hold to make the dispersion meaningful.
  - The Distance Metric $\left(d_{ij}\right)$: For every pair of locations $(i, j)$ such that $i, j \in N$, there exists a known scalar value $d_{ij}$ representing the distance between them.
    - Symmetry: In most standard PDP formulations, the distance is assumed to be symmetric, meaning $d_{ij} = d_{ji}$.
    - Reflexivity: The distance from a point to itself is zero $\left(d_{ii} = 0\right)$.
  - The Distance Matrix$(D)$: The complete set of input distances is typically organized into an $n \times n$ symmetric matrix $D$, where each entry $D_{ij}$ corresponds to $d_{ij}$. This matrix serves as the primary data source for the optimization algorithm. [1], [2]

2. Mathematical Model

The mathematical model of the Discrete p-Dispersion Problem is based on a Max-Min optimization approach, whose objective is to maximize the minimum distance between any pair of selected facilities. This approach has been employed in multiple studies addressing location problems in which proximity between facilities represents a risk or an operational disadvantage [1], [2].

- Decisions
  - $x_i \in \{0, 1\}$: A binary decision where $x_i = 1$ if candidate location $i$ is selected, and $x_i = 0$ otherwise.
  - Z: A continuos auxiliary variable representing the objective value.

- Optimization (Objective)

  Maximize Z [1], [2], [7]

To achieve this, an auxiliary variable is introduced to represent the minimum distance between the selected nodes, allowing a direct evaluation of the level of dispersion attained by a given solution [1], [5].

**Minimum dispersion constraint:**

$$z \leq d_{ij} + M\left(2 - x_i - x_j\right) \forall_i,\ j \in V,\ i < j$$

This constraint ensures that the variable z does not exceed the distance between any pair of selected nodes. When both nodes are selected, the constraint is directly activated, while the use of the parameter M allows the condition to be deactivated for pairs of nodes that are not selected, which prevents unnecessary restrictions in the model [1], [5].

**Cardinality constraint:**

$$\sum_{i=1}^{n} x_i = p$$

This constraint guarantees that exactly p locations are selected from the total set of candidate nodes, satisfying one of the fundamental requirements of the Discrete p-Dispersion Problem [1], [3].

**Integrality constraint:**

$$x_i \in \{0, 1\} \forall_i \in V$$

The binary nature of the decision variables ensures that each location is either selected or not selected in an unambiguous manner, preserving the coherence of the mathematical model [4].

**Non-negativity of the auxiliary variable:**

$$z \geq 0$$

This condition ensures that the minimum distance represented by z is non-negative, which is consistent with traditional formulations of the problem and its more recent extensions [2], [6].

Taking together, these constraints allow for a formal and consistent representation of the Discrete p-Dispersion Problem, aligning with both classical and modern formulations of the problem [1], [5], [6].

## Heuristic Description

The Greedy MaxMin algorithm is a deterministic constructive heuristic. It builds a feasible solution step-by-step by making the locally optimal choice at each stage. In the context of dispersion, it seeks to sequentially add elements that are as far away as possible from the already selected elements.

Algorithm Steps:

Let $S$ be the set of currently selected locations, and $C$ be the set of unselected candidate locations $(C = \frac{N}{S})$

1. Initialization: Select an initial pair of nodes to form $S$. To ensure a strong starting bound, the heuristic traditionally selects the two candidate nodes that are farthest apart in the entire set $N$.

   $$S = \{u, v\}\ such\ that\ d_{uv} = max\ d_{ij}\ ,\ i, j \in N$$

2. Distance Evaluation: While the number of selected facilities is less than $p(|S| < p)$, evaluate every unselected candidate $k \in C$. For each candidate, determine its distance to its closest facility that is already inside $S$.

   Let $d(k, S)$ be this shortest distance from $k$ to the current solution:

   $$d(k, S) = min\ d_{kj}\ ,\ j \in S$$

3. Greedy Selection: From the set of candidates $C$, select the node $k^*$ that maximizes the distance calculated in the previous step. This ensures the new node is placed as far away as possible from its closest neighbor in $S$.

   $$k^* = arg\ max\ d(k, S)\ ,\ k \in C$$

4. Update: Add $k^*$ to the solution set $S$ and remove it from the candidate set $C$.

   $$S = S \cup \{k^*\}$$

   $$C = C \backslash \{k^*\}$$

5. Termination: Repeat steps 2 through 4 until exactly $p$ locations have been selected ($|S| = p$). The objective value of this solution is the minimum distance between any two nodes within the final set $S$.

The Greedy MaxMin is highly efficient. Finding the initial pair takes $O(n^2)$ time. In each subsequent iteration (which runs $p - 2$ times), calculating the minimum distance for the remaining $n - |S|$ candidates requires $O(n \cdot |S|)$ operations.

Therefore, the overall time complexity of the algorithm is bounded by $O\left(n^2 + p^2 n\right)$, which simplifies to $O\left(p^2 n\right)$ or $O\left(n^2\right)$ depending on the implementation details. This makes it exceptionally fast for generating initial bounds for large-scale optimization instances.

## Problem Example

To demonstrate how the Discrete p-Dispersion Problem (MaxMinMin model) works, we can use as an example a small artificial instance representing an urban situation in which the main objective is to locate $p$ stores in $n$ rental sites available. The objective is to maximize the minimum distance between any pair of selected sites to prevent them from stealing each other's customers.

1. **Input Data (**N, p, and D**):**

   Let's assume that we have $n = 5$ candidate locations. Therefore, our candidate set is defined as: N= {1, 2, 3, 4, 5}

   And we want to open $p = 3$ facilities. This satisfies the condition of $2 \leq p < n$ ($2 \leq 3 < 5$).

   The distances between these locations are known scalars ($d_{ij}$) and are fully symmetric ($d_{ij} = d_{ji}$) with reflexivity ($d_{ii} = 0$). We organize these into a 5x5 distance matrix (D):

| D | Node 1 | Node 2 | Node 3 | Node 4 | Node 5 |
|---|---|---|---|---|---|
| Node 1 | 0 | 8 | 12 | 20 | 15 |
| Node 2 | 8 | 0 | 14 | 25 | 10 |
| Node 3 | 12 | 14 | 0 | 9 | 18 |
| Node 4 | 20 | 25 | 9 | 0 | 15 |
| Node 5 | 15 | 10 | 18 | 15 | 0 |

2. **Evaluating a Random Feasible Solution:**

   A solution is considered feasible if the exact p locations are selected, satisfying the cardinality constraint $\sum_{i=1}^{n} x_i = p$. So, let's evaluate a random subset of locations: Nodes 1, 2, and 3.

   - Our decision variables will be: $x_1 = 1$, $x_2 = 1$, $x_3 = 1$, $x_4 = 0$, $x_5 = 0$.
   - The distances between the selected pairs are:
     - $d_{12} = 8$

- $d_{13}$= 12
- $d_{23}$= 14

The objective variable Z represents the minimum distance between any of the selected nodes. For this feasible solutions, Z= min (8, 12, 14). Therefore, Z= 8

3. **Evaluating the Optimal Solution:**

   Because the MaxMinMin model aims to maximize Z , we want to find a set of 3 nodes that pushes the smallest pair distance as high as possible. So, let's evaluate a different feasible solution: Nodes 1, 4, and 5.

   - Our decision variables will be: $x_1$= 1, $x_4$= 1, $x_5$= 1, $x_2$= 0, $x_3$= 0.
   - The distances between the selected pairs are:
     - $d_{14}$= 20
     - $d_{15}$= 15
     - $d_{45}$= 22

   For this combination , the objective function yields Z= min (20, 15, 22). Therefore, Z= 15.

   Comparing the two feasible solutions, locating the facilities at {1, 4, 5} yields a greater minimum dispersion (Z=15) than locating them at {1, 2, 3} (Z=8). The optimization algorithm's goal is to systematically locate this optimal subset.

This example demonstrates how the MaxMinMin formulation transforms the dispersion objective into a structured optimization problem, where the best solution is the one that maximizes the minimum pairwise distance.

4. **Evaluating a Different Strategy**

For the same example, we can use a different strategy that looks for the same goal as the MaxMin model, being called as a sequential strategy, by using the same given example, we evaluate a subset of $p$ locations, in this case 3, where we look first for the two nodes that are more apart from each other. Picking node 2 and 4: Z= { 2, 4}, having $d_{24}$= 25.

As stated the heuristic looks to Maximize the smallest pair distance, so we evaluate all the nodes distance to the ones in the set, choosing the node that Maximizes its minimum distance, choosing node 5, that has the MAX minimum distance between the nodes chosen. Meaning that for this example by using the Sequential Strategy the objective function yields as: Z= { 2, 4, 5}; $d_{24}$= 25, $d_{42}$= 25, $d_{25}$= 10

Z= min { 25, 25, 10}. Therefore Z= 10

## Local Search

Local Search (LS) is an iterative heuristic improvement procedure designed to refine an existing feasible solution, typically generated by a constructive heuristic, to maximize the minimum pairwise distance (dispersion). The local search heuristic operates by evaluating **SWAP** movements—removing one selected node and introducing one unselected node—to determine if the minimum dispersion value improves. The search employed the **First Found method**. This means that the search accepts the very first neighboring solution encountered that results in an improved objective function value, and then proceeds from that new point, rather than evaluating the entire neighborhood for the absolute best move.

## SWAP Example

To show how the heuristic improvement procedures work in the Discrete p-Dispersion Problem, we can analyze a Local Search (LS) example using the SWAP operator. Unlike constructive heuristics, which generate a feasible solution sequentially, Local Search attempts to improve an existing configuration by exploring neighboring solutions. The objective remains the same: maximize the minimum pairwise distance between the selected nodes.

1. **Initial Feasible Solution and Motivation for Improvement:**
   Suppose that a constructive heuristic has already selected the following subset of nodes:

- Selected nodes = {9, 19, 12}

  For this configuration, the minimum pairwise distance between the selected nodes is: Minimum pairwise distance = 4.35

Although this solution is feasible because the exact number of required nodes remains selected, the constructive heuristic may have produced a suboptimal configuration due to greedy sequential decisions. Therefore, a Local Search procedure is applied to explore neighboring solutions through SWAP movements.

**2. Applying the SWAP Operator:**

A SWAP movement consists of removing one selected node and replacing it with one unselected node. The resulting configuration is evaluated to determine whether the minimum dispersion improves.

- Remove node 12
- Add node 6

After performing the SWAP movement, the new selected subset becomes:

- Selected nodes = {9, 19, 6}

The minimum pairwise distance for this new configuration is recalculated:

- Minimum pairwise distance = 4.92

Since the new minimum dispersion value is greater than the previous one (4.92 > 4.35), the SWAP movement is accepted as an improvement.

**3. Feasibility and Acceptance Criteria:**

During the Local Search process, every SWAP movement must satisfy the following conditions:

- Selection Count: Exactly p nodes must remain selected after the exchange.
- Distance Recalculation: All pairwise distances for the new configuration must be recomputed.
- Improvement Condition: The SWAP is accepted only if the minimum pairwise distance increases.

This example illustrates how Local Search improves an existing feasible solution by systematically exploring neighboring configurations. The process continues iteratively until no additional SWAP movement can further improve the minimum dispersion value, reaching convergence.

## Computational Results

The purpose of the Computational Results section is to implement and evaluate our selected strategies across various instances to determine the efficiency of the MaxMin heuristic for the P-Dispersion Problem (PDP).

The primary objective of comparing these two strategies is to assess how a minor modification in the initial iteration impacts the final value of the objective function. To analyze the behavior of both approaches, we considered different values for the total number of candidate locations (n) and the number of facilities to be selected (p). Specifically, we evaluated combinations of n = 1,000, 10,000, and 100,000, with corresponding p values of 32, 100, and 316 (where p is calculated as the square root of n). For each combination, we simulated 20 independent instances.

The algorithms were implemented in Python, utilizing the pandas, NumPy, and time libraries. Pandas and NumPy were essential for efficiently processing large-scale numerical datasets and optimizing the performance of the heuristics. Furthermore, to accurately measure and compare the computational performance of each strategy, the time library was utilized to record the total CPU execution time (in seconds) required to solve each instance.

Finally, to visually illustrate which strategy performs best, we aggregated the computational results into graphical representations. These compare the objective function values achieved by each strategy against their respective CPU execution times.

## EXPERIMENT 1: S1CH vs S2CH(No local search)

**SMALL INSTANCES:**

| | | SET S: n=1,000 | | | | Value Improvement | |
|---|---|---|---|---|---|---|---|
| **Instance** | **P** | **CH_S1** | **S1_time (cpu sec)** | **CH_S2** | **S2_time (cpu sec)** | **IMP of S1 over S2** | **IMP of S2 over S1** |
| data1000_1 | 32 | 15,6655 | 0,001 | 15,5614 | 0,002 | 0,006689629468 | -0,006645175705 |
| data1000_2 | 32 | 15,8116 | 0,001 | 16,3718 | 0,002 | -0,03421737378 | 0,03542968454 |
| data1000_3 | 32 | 16,144 | 0,001 | 16,4416 | 0,002 | -0,01810042818 | 0,01843409316 |
| data1000_4 | 32 | 15,9425 | 0,001 | 16,5774 | 0,002 | -0,03829913014 | 0,03982436883 |
| data1000_5 | 32 | 16,1741 | 0,001 | 17,037 | 0,002 | -0,05064858837 | 0,0533507274 |
| data1000_6 | 32 | 15,5002 | 0,001 | 16,4841 | 0,002 | -0,05968782038 | 0,0634766003 |
| data1000_7 | 32 | 15,6359 | 0,001 | 15,9638 | 0,002 | -0,02054022225 | 0,02097097065 |
| data1000_8 | 32 | 15,6413 | 0,001 | 16,5897 | 0,002 | -0,05716800183 | 0,06063434625 |
| data1000_9 | 32 | 15,5597 | 0,001 | 16,0347 | 0,002 | -0,02962325457 | 0,03052758087 |
| data1000_10 | 32 | 15,6617 | 0,001 | 15,7815 | 0,002 | -0,007591166873 | 0,00764923348 |
| data1000_11 | 32 | 15,5464 | 0,001 | 16,4921 | 0,002 | -0,05734260646 | 0,06083080327 |
| data1000_12 | 32 | 15,8841 | 0,001 | 16,1601 | 0,002 | -0,01707910223 | 0,01737586643 |
| data1000_13 | 32 | 16,2201 | 0,001 | 16,6984 | 0,002 | -0,02864346285 | 0,02948810427 |
| data1000_14 | 32 | 16,0592 | 0,001 | 15,8275 | 0,002 | 0,01463907755 | -0,01442786689 |
| data1000_15 | 32 | 16,0345 | 0,001 | 16,6936 | 0,002 | -0,03948219677 | 0,04110511709 |
| data1000_16 | 32 | 15,3094 | 0,001 | 16,0077 | 0,002 | -0,04362275655 | 0,04561249951 |
| data1000_17 | 32 | 15,1443 | 0,001 | 16,0355 | 0,003 | -0,05557668922 | 0,05884722305 |
| data1000_18 | 32 | 15,4551 | 0,001 | 16,611 | 0,002 | -0,06958641864 | 0,07479084574 |
| data1000_19 | 32 | 15,6972 | 0,001 | 16,0508 | 0,002 | -0,02203005458 | 0,02252631042 |
| data1000_20 | 32 | 15,605 | 0,001 | 16,2032 | 0,002 | -0,03691863336 | 0,03833386735 |
| | | | | | **Average Improve** | **-3,32%** | **3,49%** |

Table 1.Comparison of Small set between the Sequential and Random Strategy

Table 1 presents a comparison between the two selected strategies for the MaxMin Heuristic: CH_S1 and CH_S2 (Random Strategy). The fundamental difference between the two lies in their initial iteration. The First Strategy deterministically selects the two nodes that are farthest apart from each other, whereas the Random Strategy selects a random initial node and subsequently pairs it with the node farthest from it.

In the comparison of small instances (n = 1,000 and p = 32), CH_S2 outperforms CH_S1 in 19 out of 20 instances, achieving an average improvement of 3.49% in the objective function value. The sole exception is instance data1000_14, where S1 marginally outperformed S2 by 1.46%. This anomaly is likely due to S1's deterministic starting configuration coincidentally aligning with a near-optimal initial node arrangement for that specific dataset.

The average improvement of S1 over S2 is -3.32%, confirming that SI consistently underperforms relative to S2. This outcome aligns statistically with the expected behavior when comparing a single-start deterministic greedy algorithm to a multi-start randomized approach; the latter inherently possesses a higher probability of avoiding suboptimal initial configurations.

Regarding computational cost, both heuristics operate within the same order of magnitude, with S1 requiring approximately 0.001 seconds and S2 requiring 0.002 seconds per instance. This two-fold increase in execution time for S2 is entirely justified by its five independent restarts. Ultimately, this demonstrates that S2 delivers a 3.49% gain in solution quality at an essentially negligible additional computational cost for instances of this size.

**MEDIUM INSTANCES:**

| | | SET M: n=10,000 | | | | Value Improvement | |
|---|---|---|---|---|---|---|---|
| **Instance** | **P** | **CH_S1** | **S1_time (cpu sec)** | **CH_S2** | **S2_time (cpu sec)** | **IMP of S1 over S2** | **IMP of S2 over S1** |
| data1000_1 | 100 | 8,4553 | 0,003 | 8,8524 | 0,014 | -0,04485789165 | 0,04696462574 |
| data1000_2 | 100 | 8,4504 | 0,003 | 8,8749 | 0,013 | -0,04783152486 | 0,05023430844 |
| data1000_3 | 100 | 8,3785 | 0,003 | 9,139 | 0,014 | -0,08321479374 | 0,09076803724 |
| data1000_4 | 100 | 8,3568 | 0,003 | 8,7661 | 0,013 | -0,04669123099 | 0,04897807773 |
| data1000_5 | 100 | 8,3573 | 0,003 | 8,3688 | 0,014 | -0,001374151611 | 0,001376042502 |
| data1000_6 | 100 | 8,3714 | 0,003 | 8,623 | 0,015 | -0,02917778035 | 0,03005471008 |
| data1000_7 | 100 | 8,4589 | 0,003 | 8,69 | 0,014 | -0,02659378596 | 0,02732033716 |
| data1000_8 | 100 | 8,4098 | 0,003 | 8,6915 | 0,014 | -0,03241097624 | 0,03349663488 |
| data1000_9 | 100 | 8,4292 | 0,003 | 8,9412 | 0,014 | -0,0572630072 | 0,06074123286 |
| data1000_10 | 100 | 8,3967 | 0,003 | 8,8151 | 0,013 | -0,04746401062 | 0,04982909953 |
| data1000_11 | 100 | 8,3719 | 0,003 | 8,7741 | 0,014 | -0,04583945932 | 0,04804166318 |
| data1000_12 | 100 | 8,4285 | 0,003 | 8,6864 | 0,013 | -0,02969009026 | 0,03059856439 |
| data1000_13 | 100 | 8,3705 | 0,003 | 8,9805 | 0,014 | -0,0679249485 | 0,0728749776 |
| data1000_14 | 100 | 8,3659 | 0,003 | 8,6762 | 0,014 | -0,0357645052 | 0,03709104818 |
| data1000_15 | 100 | 8,361 | 0,003 | 8,9443 | 0,014 | -0,06521471775 | 0,06976438225 |
| data1000_16 | 100 | 8,3063 | 0,003 | 8,6024 | 0,014 | -0,0344206268 | 0,03564764095 |
| data1000_17 | 100 | 8,3528 | 0,003 | 8,575 | 0,014 | -0,02591253644 | 0,02660185806 |
| data1000_18 | 100 | 8,4034 | 0,003 | 8,8426 | 0,014 | -0,04966864949 | 0,05226455958 |
| data1000_19 | 100 | 8,2608 | 0,003 | 8,9775 | 0,013 | -0,07983291562 | 0,08675915166 |
| data1000_20 | 100 | 8,3873 | 0,002 | 9,0954 | 0,013 | -0,07785254084 | 0,084425262 |
| | | | | | **Average Improve** | **-4,65%** | **4,92%** |

Table 2.Comparison of Medium set between the Sequential and Random Strategy

Table 8 shows the medium instances (n=10,000 and p=100)comparison , CH_S2 (Random Strategy of the GMM) outperforms CH_S1 in all 20 instances. This reaffirms that the Random Strategy performs better on average than the First Strategy, achieving an average improvement of 4.92% over the first

strategy. Conversely, the average improvement of the First Strategy over the Random Strategy is -4.65%, clearly demonstrating that S1 overall underperforms compared to S2.

Furthermore, as the instance size increases, the average improvement of S2 over S1 also grows. It shifts from 3.49% in the small instances to 4.92% in the medium instances, representing a comparative increase of 1.43 percentage points.

Another important metric to consider is the computational cost of both strategies. Although the second strategy requires more execution time than the first, it remains well under one CPU second per instance. This efficiency confirms that S2 is still a highly feasible and superior alternative to Strategy 1.

**LARGE INSTANCES:**

| | | SET L: n=10,000 | | | | Value Improvement | |
|---|---|---|---|---|---|---|---|
| **Instance** | **P** | **CH_S1** | **S1_time (cpu sec)** | **CH_S2** | **S2_time (cpu sec)** | **IMP of S1 over S2** | **IMP of S2 over S1** |
| data1000_1 | 316 | 4,3584 | 0,066 | 4,816 | 0,327 | -0,0950166113 | 0,1049926579 |
| data1000_2 | 316 | 4,3479 | 0,075 | 4,937 | 0,334 | -0,1193234758 | 0,1354906967 |
| data1000_3 | 316 | 4,3501 | 0,078 | 4,9633 | 0,371 | -0,1235468338 | 0,1409622767 |
| data1000_4 | 316 | 4,3512 | 0,071 | 4,7692 | 0,342 | -0,08764572675 | 0,09606545321 |
| data1000_5 | 316 | 4,3553 | 0,073 | 5,0432 | 0,371 | -0,1364014911 | 0,1579454917 |
| data1000_6 | 316 | 4,3428 | 0,072 | 4,7099 | 0,355 | -0,07794220684 | 0,08453071751 |
| data1000_7 | 316 | 4,3657 | 0,069 | 4,8679 | 0,341 | -0,1031656361 | 0,1150330989 |
| data1000_8 | 316 | 4,3603 | 0,074 | 4,8002 | 0,362 | -0,09164201492 | 0,1008875536 |
| data1000_9 | 316 | 4,3492 | 0,069 | 4,7894 | 0,387 | -0,09191130413 | 0,1012140164 |
| data1000_10 | 316 | 4,3499 | 0,072 | 4,7818 | 0,339 | -0,0903216362 | 0,09928963884 |
| data1000_11 | 316 | 4,3502 | 0,075 | 4,8284 | 0,392 | -0,09903901914 | 0,1099259804 |
| data1000_12 | 316 | 4,349 | 0,085 | 4,7738 | 0,347 | -0,08898571369 | 0,09767762704 |
| data1000_13 | 316 | 4,3659 | 0,077 | 5,004 | 0,361 | -0,1275179856 | 0,1461554319 |
| data1000_14 | 316 | 4,354 | 0,072 | 4,8244 | 0,344 | -0,09750435287 | 0,1080385852 |
| data1000_15 | 316 | 4,3402 | 0,071 | 4,7347 | 0,342 | -0,08332101295 | 0,09089442883 |
| data1000_16 | 316 | 4,3505 | 0,077 | 4,892 | 0,35 | -0,110690924 | 0,1244684519 |
| data1000_17 | 316 | 4,361 | 0,068 | 4,9586 | 0,365 | -0,1205178881 | 0,1370327906 |
| data1000_18 | 316 | 4,3438 | 0,068 | 4,7193 | 0,384 | -0,07956688492 | 0,08644504811 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| data1000_19 | 316 | 4,3605 | 0,067 | 4,8348 | 0,375 | -0,09810126582 | 0,1087719298 |
| data1000_20 | 316 | 4,3553 | 0,066 | 4,7329 | 0,339 | -0,07978195187 | 0,08669896448 |
| | | | | | **Average Improve** | **-10,01%** | **11,16%** |

Table 3.Comparison of Large set between the Sequential and Random Strategy

Finally, Table 3 presents the comparative results for the large instances (n = 100,000 and p = 316). Consistent with the trends observed at smaller scales, the average improvement of CH_S2 over CH_S1 continues to escalate. The improvement grows from 4.92% in the medium instances to 11.16% in the large instances, representing an additional gain of 6.24 percentage points.

Regarding computational cost, CH_S2 requires significantly more processing effort, reflected by an average time difference of -395.15% relative to CH_S1. Nevertheless, because the absolute CPU execution time remains under one second, CH_S2 retains its practical feasibility while delivering superior results.

**AVERAGE IMPROVEMENTS:**

| **Average Value Improvements** | **IMP of S1 over S2** | **IMP of S2 over S1** |
|---|---|---|
| SET S: n=1,000 | **-3.32%** | **3.49%** |
| SET M: n=10,000 | **-4.65%** | **4.92%** |
| SET L: n=100,000 | **-10.01%** | **11.16%** |

Table 4.Comparison of Average Value Improvements between the Heuristics strategies

Table 4 presents a summary of the relative improvements between the two strategies. The data demonstrates that as the instance size increases, S2 consistently achieves greater improvements over S1. Conversely, S1 yields its poorest results in the largest datasets, clearly indicating that the performance of S2 scales significantly better with problem size. Consequently, we can conclude that the Random strategy (S2) produces superior results in problems featuring a large number of candidate nodes, making it the most effective approach to apply within the MaxMin Heuristic.

| Average Time Improvements | IMP of S1 over S2 | IMP of S2 over S1 |
|---|---|---|
| SET S: n=1,000 | **50.83%** | **-105.00%** |
| SET M: n=10,000 | **78.53%** | **-369.17%** |
| SET L: n=100,000 | **79.68%** | **-395.15%** |

Table 5.Comparison of Average Time Improvements between the Heuristics strategies

Table 5 compares the computational time required by each strategy to generate a solution. The results indicate that, on average, CH_S1 executes much faster than CH_S2. However, as demonstrated in the previous comparative tables, the execution times for both strategies remain well under a single CPU second. Because both methods evaluate so rapidly, the speed advantage of Strategy 1 is practically negligible.

In contrast, the improvements in the objective function value provided by Strategy 2 have a major impact on the final decision. Therefore, despite the lower computational cost of Strategy 1, its inferior performance regarding solution quality makes it difficult to justify. Consequently, Strategy 2 remains the superior choice for implementing the MaxMin heuristic for the P-Dispersion Problem.

## SWAP Computational Results

### EXPERIMENT 2: S1CH vs LSCHS1

**SMALL INSTANCES:**

| | | SET S: n=1,000 | | | | Value Improvement | |
|---|---|---|---|---|---|---|---|
| **Instance** | **P** | **CH_S1** | **S1_time (cpu sec)** | **LS_S1** | **LS_time (cpu sec)** | **Absolute Imp** | **Rel Improvement** |
| data1000_1 | 32 | 15,6655 | 0,001 | 16,0514 | 0,75 | 0,3859 | 2,46% |
| data1000_2 | 32 | 15,8116 | 0,001 | 16,0312 | 0,006 | 0,2196 | 1,39% |
| data1000_3 | 32 | 16,144 | 0,001 | 16,1925 | 0,002 | 0,0485 | 0,30% |
| data1000_4 | 32 | 15,9425 | 0,001 | 16,1279 | 0,003 | 0,1854 | 1,16% |
| data1000_5 | 32 | 16,1741 | 0,001 | 16,5583 | 0,005 | 0,3842 | 2,38% |
| data1000_6 | 32 | 15,5002 | 0,001 | 16,3394 | 0,01 | 0,8392 | 5,41% |
| data1000_7 | 32 | 15,6359 | 0,001 | 16,3161 | 0,008 | 0,6802 | 4,35% |
| data1000_8 | 32 | 15,6413 | 0,001 | 15,7465 | 0,002 | 0,1052 | 0,67% |
| data1000_9 | 32 | 15,5597 | 0,001 | 15,9044 | 0,007 | 0,3447 | 2,22% |
| data1000_10 | 32 | 15,6617 | 0,001 | 16,657 | 0,015 | 0,9953 | 6,35% |
| data1000_11 | 32 | 15,5464 | 0,001 | 15,6038 | 0,002 | 0,0574 | 0,37% |
| data1000_12 | 32 | 15,8841 | 0,001 | 16,388 | 0,008 | 0,5039 | 3,17% |
| data1000_13 | 32 | 16,2201 | 0,001 | 16,2201 | 0,003 | 0 | 0,00% |
| data1000_14 | 32 | 16,0592 | 0,001 | 16,331 | 0,012 | 0,2718 | 1,69% |
| data1000_15 | 32 | 16,0345 | 0,001 | 16,0345 | 0,002 | 0 | 0,00% |
| data1000_16 | 32 | 15,3094 | 0,001 | 16,781 | 0,021 | 1,4716 | 9,61% |
| data1000_17 | 32 | 15,1443 | 0,001 | 15,1587 | 0,004 | 0,0144 | 0,10% |
| data1000_18 | 32 | 15,4551 | 0,001 | 16,6521 | 0,013 | 1,197 | 7,75% |
| data1000_19 | 32 | 15,6972 | 0,001 | 16,6025 | 0,01 | 0,9053 | 5,77% |
| data1000_20 | 32 | 15,605 | 0,001 | 17,2467 | 0,02 | 1,6417 | 10,52% |
| | | | | | | **Average Improve** | **3,28%** |

Table 6.Comparison of Strategy 1 vs Local Search. Small size Set

The Local Search consistently improved upon the S1 constructive solution across 18 out of 20 instances, with an average relative improvement of 3.28%. The two exceptions — data1000_13 and data1000_15 — show zero improvement, indicating that the SI solution was already at a local optimum for those particular instances, meaning no beneficial swap could be found from that starting configuration.

The improvement is notably non-uniform across instances, ranging from 0.10% (data1000_17) up to 10.52% (data1000_20). This variance suggests that the quality of the SI constructive solution directly impacts how much room the Local Search has to improve — instances where S1 starts from a poor configuration benefit significantly more from the neighborhood search.

Regarding computational cost, the LS runtime ranges from 0.002s to 0.75s, which is considerably more variable than the SI constructive phase (fixed at 0.001s). This is expected behavior for a first-found Local Search — instances requiring more swap iterations before reaching a local optimum naturally take longer to converge.

**MEDIUM INSTANCES:**

| | | **SET M: n=10,000** | | | | **Value Improvement** | |
|---|---|---|---|---|---|---|---|
| **Instance** | **P** | **CH_S1** | **S1_time (cpu sec)** | **LS_S1** | **LS_time (cpu sec)** | **Absolute Imp** | **Rel Improvement** |
| data1000_1 | 100 | 8,4553 | 0,003 | 8,9649 | 0,102 | 0,5096 | 6,03% |
| data1000_2 | 100 | 8,4504 | 0,003 | 9,2029 | 0,157 | 0,7525 | 8,90% |
| data1000_3 | 100 | 8,3785 | 0,003 | 9,0281 | 0,113 | 0,6496 | 7,75% |
| data1000_4 | 100 | 8,3568 | 0,003 | 8,4694 | 0,015 | 0,1126 | 1,35% |
| data1000_5 | 100 | 8,3573 | 0,003 | 9,0193 | 0,109 | 0,662 | 7,92% |
| data1000_6 | 100 | 8,3714 | 0,003 | 8,665 | 0,045 | 0,2936 | 3,51% |
| data1000_7 | 100 | 8,4589 | 0,003 | 8,7505 | 0,067 | 0,2916 | 3,45% |
| data1000_8 | 100 | 8,4098 | 0,003 | 9,2433 | 0,118 | 0,8335 | 9,91% |
| data1000_9 | 100 | 8,4292 | 0,003 | 8,8139 | 0,065 | 0,3847 | 4,56% |
| data1000_10 | 100 | 8,3967 | 0,003 | 9,0102 | 0,102 | 0,6135 | 7,31% |
| data1000_11 | 100 | 8,3719 | 0,003 | 8,8706 | 0,077 | 0,4987 | 5,96% |
| data1000_12 | 100 | 8,4285 | 0,003 | 9,3245 | 0,139 | 0,896 | 10,63% |
| data1000_13 | 100 | 8,3705 | 0,003 | 8,964 | 0,091 | 0,5935 | 7,09% |
| data1000_14 | 100 | 8,3659 | 0,003 | 9,1803 | 0,118 | 0,8144 | 9,73% |
| data1000_15 | 100 | 8,361 | 0,003 | 9,0955 | 0,106 | 0,7345 | 8,78% |
| data1000_16 | 100 | 8,3063 | 0,003 | 8,9164 | 0,075 | 0,6101 | 7,35% |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| data1000_17 | 100 | 8,3528 | 0,003 | 8,6873 | 0,043 | 0,3345 | 4,00% |
| data1000_18 | 100 | 8,4034 | 0,003 | 8,7848 | 0,067 | 0,3814 | 4,54% |
| data1000_19 | 100 | 8,2608 | 0,003 | 9,0148 | 0,082 | 0,754 | 9,13% |
| data1000_20 | 100 | 8,3873 | 0,002 | 9,0369 | 0,105 | 0,6496 | 7,75% |
| | | | | | | **Average Improve** | **6,78%** |

Table 7.Comparison of Strategy 1 vs Local Search. Medium size Set

Table 7 presents the impact of applying the Local Search (LS) Swap operator on the initial sequential constructive heuristic (S1) for medium-sized instances (n=10,000 and p=100 ). Unlike the small instances where in some cases there was no improvement, for this problem size, it is observed that the Local Search manages to increase the objective function value in all 20 evaluated instances. On average, a relative improvement of 6.78% was achieved over the base constructive solution.

The improvement range varies depending on the initial solution's quality, with relative gains spanning from a conservative 1.35% (instance 4) up to a significant 10.63% (instance 12). Regarding computational cost, while the S1 heuristic remains extremely fast at 0.003 CPU seconds, the LS phase logically increases execution time, fluctuating between 0.015 and 0.157 seconds based on the swaps needed to reach a local optimum. Ultimately, this fractional increase in computational effort is fully justified by the consistent enhancement in dispersion quality.

**LARGE INSTANCES:**

| | | **SET L: n=100,000** | | | | **Value Improvement** | |
|---|---|---|---|---|---|---|---|
| **Instance** | **P** | **CH_S1** | **S1_time (cpu sec)** | **LS_S1** | **LS_time (cpu sec)** | **Absolute Imp** | **Rel Improvement** |
| data1000_1 | 316 | 4,3584 | 0,066 | 4,9564 | 5,909 | 0,598 | 13,72% |
| data1000_2 | 316 | 4,3479 | 0,075 | 4,9649 | 6,039 | 0,617 | 14,19% |
| data1000_3 | 316 | 4,3501 | 0,078 | 5,1001 | 7,546 | 0,75 | 17,24% |
| data1000_4 | 316 | 4,3512 | 0,071 | 4,9494 | 5,673 | 0,5982 | 13,75% |
| data1000_5 | 316 | 4,3553 | 0,073 | 4,8699 | 5,122 | 0,5146 | 11,82% |
| data1000_6 | 316 | 4,3428 | 0,072 | 5,0588 | 7,151 | 0,716 | 16,49% |
| data1000_7 | 316 | 4,3657 | 0,069 | 5,0215 | 6,758 | 0,6558 | 15,02% |
| data1000_8 | 316 | 4,3603 | 0,074 | 4,9229 | 5,826 | 0,5626 | 12,90% |
| data1000_9 | 316 | 4,3492 | 0,069 | 4,9488 | 6,162 | 0,5996 | 13,79% |
| data1000_10 | 316 | 4,3499 | 0,072 | 5,0298 | 6,912 | 0,6799 | 15,63% |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| data1000_11 | 316 | 4,3502 | 0,075 | 4,9594 | 6,555 | 0,6092 | 14,00% |
| data1000_12 | 316 | 4,349 | 0,085 | 4,3542 | 0,07 | 0,0052 | 0,12% |
| data1000_13 | 316 | 4,3659 | 0,077 | 4,9261 | 5,505 | 0,5602 | 12,83% |
| data1000_14 | 316 | 4,354 | 0,072 | 4,354 | 0,024 | 0 | 0,00% |
| data1000_15 | 316 | 4,3402 | 0,071 | 5,0268 | 7,231 | 0,6866 | 15,82% |
| data1000_16 | 316 | 4,3505 | 0,077 | 5,0915 | 9,293 | 0,741 | 17,03% |
| data1000_17 | 316 | 4,361 | 0,068 | 4,975 | 5,944 | 0,614 | 14,08% |
| data1000_18 | 316 | 4,3438 | 0,068 | 4,8878 | 5,027 | 0,544 | 12,52% |
| data1000_19 | 316 | 4,3605 | 0,067 | 4,989 | 6,12 | 0,6285 | 14,41% |
| data1000_20 | 316 | 4,3553 | 0,066 | 4,9521 | 5,812 | 0,5968 | 13,70% |
| | | | | | | **Average Improve** | **12,95%** |

Table 8.Comparison of Strategy 1 vs Local Search. Large size Set

Table 8 presents the results of applying the Local Search procedure to the initial sequential heuristic (S1) for large-scale instances (n=100,000 and p=316). In this scenario, the Local Search demonstrates its most significant impact, achieving an impressive average relative improvement of 12.95% over the base constructive solutions. While most solutions saw significant enhancements, with instance 3 reaching a maximum gain of 17.24%, isolated cases such as instance 14 showed a 0.00% change. This demonstrates that the starting heuristic can occasionally produce a configuration that is already a local optimum.

However, this significant boost in solution quality comes at the expense of higher computational demands, which becomes much more pronounced in large-scale settings. The constructive S1 phase remains highly efficient, consistently taking under 0.09 CPU seconds. However, exploring the massive neighborhood during the Local Search phase requires significantly more effort, with execution times generally jumping to between 5 and 9.3 seconds to reach convergence. Despite this longer runtime, the nearly 13% average gain in dispersion quality confirms that Local Search is an essential refinement step for large problems.

## EXPERIMENT 3: S2CH VS LSCHS2

This experiment compares the solutions generated by S2CH against the solutions obtained after applying the Local Search procedure over S2CH. The objective is to evaluate how much the Local Search phase can improve the initial solutions generated by the constructive heuristic and analyze the trade-off between solution quality and computational time. Since Local Search explores neighbouring solutions iteratively, it is expected to obtain better objective values than the constructive heuristic alone, although with additional computational effort.

**SMALL INSTANCES:**

| | | SET S: n=1,000 | | | | Value Improvement | |
|---|---|---|---|---|---|---|---|
| **Instance** | **P** | **CH_S2** | **S2_time (cpu sec)** | **LS_S2** | **LS_time (cpu sec)** | **Absolute Imp** | **Rel Improvement** |
| data1000_1 | 32 | 15,5614 | 0,002 | 15,5614 | 1,437 | 0 | 0,00% |
| data1000_2 | 32 | 16,3718 | 0,002 | 17,0679 | 0,004 | 0,6961 | 4,25% |
| data1000_3 | 32 | 16,4416 | 0,002 | 17,1788 | 0,007 | 0,7372 | 4,48% |
| data1000_4 | 32 | 16,5774 | 0,002 | 16,8036 | 0,004 | 0,2262 | 1,36% |
| data1000_5 | 32 | 17,037 | 0,002 | 17,037 | 0,001 | 0 | 0,00% |
| data1000_6 | 32 | 16,4841 | 0,002 | 16,4841 | 0,002 | 0 | 0,00% |
| data1000_7 | 32 | 15,9638 | 0,002 | 17,1781 | 0,016 | 1,2143 | 7,61% |
| data1000_8 | 32 | 16,5897 | 0,002 | 16,8279 | 0,002 | 0,2382 | 1,44% |
| data1000_9 | 32 | 16,0347 | 0,002 | 16,0347 | 0,001 | 0 | 0,00% |
| data1000_10 | 32 | 15,7815 | 0,002 | 16,5977 | 0,01 | 0,8162 | 5,17% |
| data1000_11 | 32 | 16,4921 | 0,002 | 16,6905 | 0,003 | 0,1984 | 1,20% |
| data1000_12 | 32 | 16,1601 | 0,002 | 16,6446 | 0,006 | 0,4845 | 3,00% |
| data1000_13 | 32 | 16,6984 | 0,002 | 16,6984 | 0,001 | 0 | 0,00% |
| data1000_14 | 32 | 15,8275 | 0,002 | 16,6953 | 0,002 | 0,8678 | 5,48% |
| data1000_15 | 32 | 16,6936 | 0,002 | 16,7212 | 0,003 | 0,0276 | 0,17% |
| data1000_16 | 32 | 16,0077 | 0,002 | 16,8187 | 0,005 | 0,811 | 5,07% |
| data1000_17 | 32 | 16,0355 | 0,003 | 16,2062 | 0,002 | 0,1707 | 1,06% |
| data1000_18 | 32 | 16,611 | 0,002 | 16,611 | 0,001 | 0 | 0,00% |
| data1000_19 | 32 | 16,0508 | 0,002 | 16,8354 | 0,004 | 0,7846 | 4,89% |
| data1000_20 | 32 | 16,2032 | 0,002 | 17,3913 | 0,02 | 1,1881 | 7,33% |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | **Average Improve** | **2,63%** |

Table 9.Comparison of Strategy 2 vs  Local Search. Small size Set

This table compares the solutions generated by S2CH and the solutions improved by the Local Search procedure for small-size instances. The results show that Local Search improved the constructive solution in several experiments, obtaining better objective values in most cases. However, the improvements are smaller compared to those obtained in Experiment 2 because S2CH already generates higher-quality initial solutions during the constructive phase.

The average relative improvement remained relatively low, suggesting that many of the solutions generated by S2CH are already close to local optima. In some instances, the Local Search procedure was unable to improve the constructive solution, indicating that no beneficial SWAP movements were found during the neighborhood exploration.

Regarding computational performance, S2CH required very small execution times because it directly constructs a feasible solution without iterative refinements. In contrast, the Local Search procedure required additional CPU time due to the evaluation of neighboring solutions. Even so, the execution times remained relatively low for small-size problems.

**MEDIUM INSTANCES:**

| | | SET M: n=10,000 | | | | Value Improvement | |
|---|---|---|---|---|---|---|---|
| **Instance** | **P** | **CH_GMM** | **GMM_time (cpu sec)** | **LS_GMM** | **LS_time (cpu sec)** | **Absolute Imp** | **Rel Improvement** |
| data1000_1 | 100 | 8,8524 | 0,014 | 9,0704 | 0,014 | 0,218 | 2,46% |
| data1000_2 | 100 | 8,8749 | 0,013 | 9,3205 | 0,044 | 0,4456 | 5,02% |
| data1000_3 | 100 | 9,139 | 0,014 | 9,8465 | 0,03 | 0,7075 | 7,74% |
| data1000_4 | 100 | 8,7661 | 0,013 | 9,0675 | 0,015 | 0,3014 | 3,44% |
| data1000_5 | 100 | 8,3688 | 0,014 | 9,0722 | 0,124 | 0,7034 | 8,41% |
| data1000_6 | 100 | 8,623 | 0,015 | 9,3375 | 0,074 | 0,7145 | 8,29% |
| data1000_7 | 100 | 8,69 | 0,014 | 8,6945 | 0,006 | 0,0045 | 0,05% |
| data1000_8 | 100 | 8,6915 | 0,014 | 8,8905 | 0,022 | 0,199 | 2,29% |
| data1000_9 | 100 | 8,9412 | 0,014 | 9,0367 | 0,012 | 0,0955 | 1,07% |
| data1000_10 | 100 | 8,8151 | 0,013 | 8,9444 | 0,009 | 0,1293 | 1,47% |
| data1000_11 | 100 | 8,7741 | 0,014 | 9,2592 | 0,037 | 0,4851 | 5,53% |
| data1000_12 | 100 | 8,6864 | 0,013 | 8,7428 | 0,005 | 0,0564 | 0,65% |
| data1000_13 | 100 | 8,9805 | 0,014 | 9,8215 | 0,068 | 0,841 | 9,36% |
| data1000_14 | 100 | 8,6762 | 0,014 | 9,3026 | 0,083 | 0,6264 | 7,22% |
| data1000_15 | 100 | 8,9443 | 0,014 | 9,1564 | 0,017 | 0,2121 | 2,37% |
| data1000_16 | 100 | 8,6024 | 0,014 | 8,9548 | 0,013 | 0,3524 | 4,10% |
| data1000_17 | 100 | 8,575 | 0,014 | 8,8682 | 0,012 | 0,2932 | 3,42% |
| data1000_18 | 100 | 8,8426 | 0,014 | 9,005 | 0,021 | 0,1624 | 1,84% |
| data1000_19 | 100 | 8,9775 | 0,013 | 9,824 | 0,039 | 0,8465 | 9,43% |
| data1000_20 | 100 | 9,0954 | 0,013 | 9,1856 | 0,006 | 0,0902 | 0,99% |
| | | | | | | **Average Improve** | **4,26%** |

Table 10.Comparison of Strategy 2 vs Local Search. Medium size Set

For medium-size instances, the Local Search procedure continued improving the solutions generated by S2CH in most experiments. The results indicate that neighborhood exploration still contributes to increasing the minimum dispersion value, although the improvements remain moderate because the constructive heuristic already provides competitive initial solutions.

As the problem size increases, the search space becomes larger, allowing Local Search to identify additional beneficial exchanges between selected and non-selected nodes. However, the improvement percentages are still smaller than those observed when Local Search was applied over Strategy 1, confirming that S2CH generates stronger initial configurations.

In terms of computational time, the Local Search procedure required noticeably higher CPU times because medium-size instances contain larger neighborhoods that must be evaluated during the search process. Nevertheless, the computational effort remains acceptable considering the improvement obtained in solution quality.

**LARGE INSTANCES:**

| | | SET L: n=100,000 | | | | Value Improvement | |
|---|---|---|---|---|---|---|---|
| **Instance** | **P** | **CH_GMM** | **GMM_time (cpu sec)** | **LS_GMM** | **LS_time (cpu sec)** | **Absolute Imp** | **Rel Improvement** |
| data1000_1 | 316 | 4,816 | 0,327 | 4,8651 | 0,185 | 0,0491 | 1,02% |
| data1000_2 | 316 | 4,937 | 0,334 | 4,9991 | 0,292 | 0,0621 | 1,26% |
| data1000_3 | 316 | 4,9633 | 0,371 | 4,9968 | 0,208 | 0,0335 | 0,67% |
| data1000_4 | 316 | 4,7692 | 0,342 | 4,9413 | 0,407 | 0,1721 | 3,61% |
| data1000_5 | 316 | 5,0432 | 0,371 | 5,1216 | 0,376 | 0,0784 | 1,55% |
| data1000_6 | 316 | 4,7099 | 0,355 | 4,7587 | 0,167 | 0,0488 | 1,04% |
| data1000_7 | 316 | 4,8679 | 0,341 | 5,1688 | 3,338 | 0,3009 | 6,18% |
| data1000_8 | 316 | 4,8002 | 0,362 | 4,822 | 0,129 | 0,0218 | 0,45% |
| data1000_9 | 316 | 4,7894 | 0,387 | 4,8967 | 0,585 | 0,1073 | 2,24% |
| data1000_10 | 316 | 4,7818 | 0,339 | 4,8806 | 0,494 | 0,0988 | 2,07% |
| data1000_11 | 316 | 4,8284 | 0,392 | 4,8607 | 0,235 | 0,0323 | 0,67% |
| data1000_12 | 316 | 4,7738 | 0,347 | 4,7738 | 0,023 | 0 | 0,00% |
| data1000_13 | 316 | 5,004 | 0,361 | 5,0705 | 0,341 | 0,0665 | 1,33% |
| data1000_14 | 316 | 4,8244 | 0,344 | 4,8249 | 0,052 | 0,0005 | 0,01% |
| data1000_15 | 316 | 4,7347 | 0,342 | 5,0908 | 2,937 | 0,3561 | 7,52% |
| data1000_16 | 316 | 4,892 | 0,35 | 4,9367 | 0,209 | 0,0447 | 0,91% |
| data1000_17 | 316 | 4,9586 | 0,365 | 5,0569 | 0,433 | 0,0983 | 1,98% |
| data1000_18 | 316 | 4,7193 | 0,384 | 5,1449 | 4,724 | 0,4256 | 9,02% |
| data1000_19 | 316 | 4,8348 | 0,375 | 4,9746 | 0,882 | 0,1398 | 2,89% |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| data1000_20 | 316 | 4,7329 | 0,339 | 4,9186 | 1,441 | 0,1857 | 3,92% |
| | | | | | | **Average Improve** | **2,42%** |

Table 11.Comparison of Strategy 2 vs  Local Search. Large size Set

For large-scale instances, the Local Search procedure still obtained better objective values than S2CH in several experiments. However, the average relative improvements became smaller compared to previous experiments because the constructive heuristic already generates highly dispersed initial solutions for large problems.

The results suggest that S2CH is capable of producing solutions that are already close to strong local optima, reducing the number of profitable SWAP movements available during neighborhood exploration. Even so, Local Search continues providing additional refinement and slightly improves the final dispersion values obtained by the constructive heuristic.

The most significant difference appears in computational time. Since large-scale instances generate very large neighborhoods, the Local Search procedure requires considerably more CPU time to evaluate candidate exchanges during each iteration. Despite the increase in computational effort, the additional improvement in solution quality may justify the use of Local Search for applications where obtaining better dispersion values is more important than minimizing execution time.

**AVERAGE IMPROVEMENTS:**

| Average Value Improvements | IMP of LS over S1 | IMP of Ls over S2 |
|---|---|---|
| SET S: n=1,000 | **3.28%** | **2.63%** |
| SET M: n=10,000 | **6.78%** | **4.26%** |
| SET L: n=100,000 | **12.95%** | **2.42%** |

Table 12.Improvement of Local Search heuristic over the GMM strategies

The computational experiments demonstrate that applying Local Search over S2CH improves the quality of the constructive solutions for all tested instance sizes. However, the magnitude of the improvements is smaller compared to Experiment 2 because S2CH already generates stronger initial solutions during the constructive phase.

Although Local Search requires additional computational effort due to the neighborhood exploration process, the method remains effective for refining the solutions obtained by the constructive heuristic. Overall, the results confirm that combining constructive heuristics with Local Search procedures produces more competitive solutions for the Discrete p-Dispersion Problem.

## Conclusions

The comparative analysis of the heuristics applied to the P-Dispersion Problem revealed clear differences in both solution quality and computational behavior. In particular, the GMM (Greedy Max-Min multi-start) heuristic demonstrated a significant advantage over the SI (Sequential Insertion) heuristic by incorporating multiple random restarts. This strategy allows the exploration of different regions of the solution space, resulting in solutions that are approximately 8% to 11% better on average. In contrast, the deterministic nature of the SI approach, which relies on a single fixed starting point, tends to limit its ability to escape lower-quality solutions.

Additionally, an important effect related to problem density was observed. As the number of nodes increases from 1,000 to 100,000 within the same spatial domain, the minimum distance between selected points decreases significantly, from approximately 16 to 4.8 units. This behavior reflects an inherent limitation of the problem: as more nodes are packed into a fixed area, maintaining high dispersion becomes increasingly difficult, directly impacting the objective function value.

Moreover, the application of Local Search proved to be a highly effective strategy for improving initial solutions generated by constructive heuristics. In all tested instances, this method was able to maintain or enhance solution quality, with the most notable improvements observed in medium-sized instances (e.g., around n = 10,000), achieving average gains of approximately 4.26%. This confirms that while constructive heuristics are useful for quickly generating feasible solutions, their combination with refinement techniques is essential for obtaining more competitive results.

Finally, it was identified that the computational cost associated with distance calculations becomes the main bottleneck in large-scale instances. In particular, for n = 100,000, GPU acceleration was required to maintain reasonable execution times. This finding highlights the importance of considering not only solution quality but also computational efficiency and available resources when implementing optimization algorithms for large-scale problems.

## References

[1] Erkut, E. (1990). The discrete p-dispersion problem. European Journal of Operational Research.

[2] Erkut, E., & Neuman, S. (1991). Comparison of four models for dispersing facilities. INFOR: Information Systems and Operational Research.

[3] Erkut, E., Ulkusal, Y., & Yenicerioglu, O. (1994). A comparison of simple search and greedy heuristics for the p-dispersion problem. Computers & Operations Research.

[4] Pisinger, D. (2006). Upper bounds and exact algorithms for the maximum dispersion problem. Computers & Operations Research.

[5] Sayyady, F., & Saboonchi, H. (2018). A New Compact Formulation for the Discrete p-Dispersion Problem. European Journal of Operational Research.

[6] Saboonchi, H., et al. (2022). MaxMinMin p-dispersion problem. Computers & Operations Research.

[7] B. Saboonchi, P. Hansen, P. Sylvain (2013). MaxMinMin p-dispersion problem: A Variable Neighborhood Search Approach. Computers & Operations Research