

Article

Exploring the Impact of Local Operator Configurations in the Multi-Demand Multidimensional Knapsack Problem

José García * , Ivo Cattarinich *, Paola Moraga  and Hernan Pinto 

Escuela de Ingeniería de Construcción y Transporte, Pontificia Universidad Católica de Valparaíso, Valparaíso 2362804, Chile; paola.moraga@pucv.cl (P.M.); hernan.pinto@pucv.cl (H.P.)

* Correspondence: jose.garcia@pucv.cl (J.G.); ivo.cattarinich.g@mail.pucv.cl (I.C.)

Abstract: The Multi-demand Multidimensional Knapsack Problem (MDMKP) is a challenging combinatorial task due to its capacity and demand constraints. Local search operators play a key role in metaheuristics when navigating such complex solution spaces, yet their impact on MDMKP performance has received limited attention. In this work, we investigate four local operator configurations—Add, Drop, Swap, and All Operator—within the Whale Optimization Algorithm framework. Our approach integrates these operators to broaden search coverage and refine candidate solutions. This design aims to enhance solution quality by balancing exploration and exploitation across multiple dimensions of the MDMKP. Experimental results on benchmark instances with different sizes ($n = 100, 250$, and 500) show that the All Operator configuration consistently achieves better maximum and average values. In large-scale instances ($n = 500$), the “All Operator” configuration achieves an average maximum value of 107,967, which is approximately 1.4% higher than the 106,490 achieved by the “Add Operator” and about 0.2% higher than the 107,771 obtained by the “Swap Operator”, while significantly outperforming the “Drop Operator” (average maximum of 99,164). Statistical tests confirm its advantage over the other configurations, suggesting that combining multiple local operators can significantly strengthen performance in high-dimensional and constraint-heavy settings like the MDMKP.



Academic Editors: Stylianos Pappas, Georgios Fotis and Vasiliki Vita

Received: 6 January 2025

Revised: 9 February 2025

Accepted: 14 February 2025

Published: 16 February 2025

Citation: García, J.; Cattarinich, I.; Moraga, P.; Pinto, H. Exploring the Impact of Local Operator Configurations in the Multi-Demand Multidimensional Knapsack Problem. *Appl. Sci.* **2025**, *15*, 2059. <https://doi.org/10.3390/app15042059>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: combinatorial optimization; machine learning; metaheuristics; multi-demand multidimensional knapsack problem

1. Introduction

In recent years, the effectiveness of metaheuristics in addressing complex challenges, especially those involving intricate combinatorial problems, has been established. This utility is evidenced across various fields, such as mechanical engineering [1], logistics [2], civil engineering [3,4], and operations research [5], among others. The pursuit of more efficient metaheuristic algorithms continues to be an important area of research. This is particularly true as optimization problems, especially combinatorial or mixed types, grow in complexity with the addition of new variables and constraints. Additionally, the non-free lunch theorem [6] emphasizes that no single optimization technique can consistently outperform all others across every possible problem space. A promising direction for the advancement of metaheuristic algorithms is seen in the strategic integration of various techniques.

The method of hybridization, which involves the integration of various metaheuristic algorithms to improve performance, stands as a notable strategy in the realm of metaheuristic combinations. An example of this approach is the development of an algorithm that

utilizes the hybridization of two metaheuristics for addressing the feature selection issue, as discussed in [7]. This study introduced the application of feature selection for IoT Intrusion Detection, evaluating the efficiency of the proposed hybrid algorithm by comparing its outcomes with those obtained from the original methodologies. The enhanced performance of the hybrid algorithm was evident in both the quality of solutions and the acceleration of convergence times. Additionally, ref. [8] presented another persuasive instance where the challenge of extending the lifespan of wireless sensor networks through cluster generation techniques was explored. Here, a hybrid model was proposed by the authors, merging the particle swarm optimization algorithm with the Sea Lion technique for the purpose of cluster identification. When compared to traditional algorithms like the genetic algorithm and particle swarm optimization, this hybridized approach demonstrated a superior performance, surpassing these methods by more than 11% in solution quality.

Exploring the integration of additional operators into metaheuristics has emerged as a significant field of study. The enhancement of algorithmic performance through various strategies, including the use of initialization operators, local search mechanisms, and binarization processes, has gained attention in scholarly work for its effectiveness in boosting algorithmic outcomes. The strategic employment of initialization operators is pivotal in defining the search space boundaries, thereby enabling more effective exploration that may yield superior outcomes. Local search mechanisms are essential in circumventing local optima and guiding the search process towards more promising areas by iteratively refining potential solutions. Moreover, the application of binarization processes facilitates the transformation of continuous problems into their discrete counterparts, expanding the metaheuristic algorithms' range of applicability to include a wider variety of problem types. Consequently, these additional operators are lauded for their instrumental role in enhancing the flexibility and overall performance of metaheuristic approaches.

In this context, an exploration of these aspects has been delved into by some works. For instance, an in-depth examination of the influence of various solution initialization methods on the effectiveness of a hybrid algorithm, tailored for the set union knapsack problem, was conducted in the research outlined in [9]. An unequivocal connection between the initialization strategy and the algorithm's performance was conclusively established by this study, with the supremacy of the weighted method being persistently marked. Notably, both self-sufficiency and remarkable competitiveness were reflected in the study's outcomes when benchmarked against other established metaheuristic methodologies. The application of local search operators was analyzed in [10], where the study assessed the enhancement potential in the performance of three metaheuristics (Iterated Greedy Search-IGS, Artificial Bee Colony-ABC, Genetic Algorithm-GA) when the local search operator was replaced by Variable Neighborhood Descent (VND). Through various tests and analyses, this hypothesis was validated by the study, revealing that the integration of VND generally improved the performance of metaheuristics and yielded results that were equivalent to or better than the state-of-the-art approaches.

Building upon the research strands previously outlined, this article undertakes an exploration into the performance of various configurations of a local operator, applied to the Multi-Demand Multidimensional Knapsack Problem (MDMKP). The classic Multidimensional Knapsack Problem (MKP) represents a specific instance within the broader context of MDMKP. Characterized by its stringent capacity and demand constraints, the MDMKP poses a significant challenge, where even achieving a valid solution constitutes a substantial obstacle. This article contributes the following:

1. The MDMKP is addressed through the application of hybrid techniques, integrating unsupervised learning methodologies and the optimization technique known as the Whale Optimization Algorithm (WoA) [11].

2. An analysis was conducted on the impact resulting from the incorporation of a local operator and the exploration of four configurations thereof on the outcomes of the various instances of MDMKP.

This paper is structured as follows: In Section 2, we provide an in-depth discussion of the MDMKP problem and its respective applications. An explanation of the binary algorithm and the local operator is offered in Section 3. Detailed numerical experiments and comparative analyses are presented in Section 4. The paper concludes in Section 5, where conclusions are drawn and potential directions for future research are suggested.

2. The Multi-Demand Multidimensional Knapsack Problem

In this section, the formulation of the MDMKP is provided, followed by a focused review of prior studies relevant to the techniques employed in this work. The MDMKP is an expanded version of the Multidimensional Knapsack Problem (MKP). It is formulated as follows: Let $V = \{1, \dots, n\}$ represent the collection of items, and $C = \{c_1, \dots, c_m, \dots, c_{m+q}\}$ constitute the set of constraints. A profit p_j is associated with each item. The objective is to maximize z , as shown in Equation (1), where x_j is 1 if item j is selected and 0 otherwise. Constraints are defined by m inequalities in Equation (2) and q inequalities in Equation (3):

$$\text{Maximize } z = \sum_{j=1}^n p_j x_j \quad (1)$$

$$\text{subject to } \sum_{j=1}^n a_{ij} x_j \leq c_i, \quad \forall i \in \{1, \dots, m\} \quad (2)$$

$$\sum_{j=1}^n a_{ij} x_j \geq c_i, \quad \forall i \in \{m+1, \dots, m+q\} \quad (3)$$

Distinct from the MKP, the MDMKP permits values of p_i to be less than, equal to, or greater than zero, and also incorporates the greater-than-or-equal constraints found in Inequality (3). The inequalities present in Equation (2) are identified as knapsack constraints, whereas those present in Equation (3) are acknowledged as demand constraints.

The MDMKP has been observed to garner less attention than the MKP, potentially due to its two types of constraints which complicate the identification of feasible solutions. Both exact and heuristic methods have been explored in the literature. For example, general mixed-integer programming solvers such as CPLEX have been effectively employed to solve instances with $n \leq 100$ within a reasonable time span [12,13]. However, for larger instances, the existing procedures often falter in pinpointing an optimal solution. Consequently, various heuristic approaches have been introduced to tackle large instances.

In [14], a two-stage tabu search algorithm is employed where neighborhoods with diverse structures are utilized. Initially, a tabu search is conducted to explore infeasible regions. Following the establishment of feasibility, a second tabu operation is executed to solely scrutinize feasible solutions. A straightforward iterative search strategy was implemented in [15] to resolve the MDMKP. This strategy employed the adaptive memory and learning techniques derived from the tabu search. The search hinged on a strategic oscillation around feasibility boundaries, orchestrating the interaction between the alterations in objective function values and the changes in feasibility. Subsequently, standard aspiration criteria and dynamic short-term tabu tenure were applied.

In [16], the authors developed a black-box scatter search approach for general categories of binary optimization problems and evaluated their method using the MDMKP, amongst other binary problems. Additionally, the solution addressed MDMKP constraints through a static penalty approach. A solution-based tabu search method was investigated in [12]. This algorithm was designed in two unique stages. The first stage involved identi-

finding a promising hyperplane within the entire search space, and the second stage aimed to discover improved solutions by probing the subspace of the hyperplane. Moreover, the key components of the algorithm were evaluated to discern their impacts on the algorithm's performance.

In [17], a hybrid algorithm was presented that combined the k-means method with binary versions of the cuckoo search and sine cosine algorithms to solve the NP-hard MDMKP. The effectiveness of the algorithm was proven through extensive experiments, which demonstrated its superior performance over other binarization methods. In particular, integrating a repair operator improved results by an average of 0.23%, while the k-means operator outperformed the random operator by an average of 1.06% for cases with 100 items and 30–30 restrictions.

The MDMKP has numerous real-world applications across various domains. For example, ref. [18] examined the dual challenge of locating obnoxious activities and routing hazardous materials between constructed areas and facilities. Such facilities, which pose significant risks to human health and the environment, include landfills, chemical plants, electrical power distribution networks, and nuclear power plants. Similarly, ref. [19] introduced a mathematical formulation based on the Multi-Demand Multidimensional Knapsack Problem, designed to assist in the selection of projects for R&D portfolios. This approach ensured adherence to diverse constraints, such as budget, personnel availability, and strategic goals, underscoring the versatility of the MDMKP in addressing complex decision-making scenarios.

An essential component of solving combinatorial optimization problems is the effective use of local search operators. These operators refine candidate solutions and intensify the search within promising regions of the solution space. For instance, ref. [20] employed a multi-start iterated local search algorithm to address the generalized quadratic multiple knapsack problem (G-QMKP). By integrating techniques like VND, the algorithm systematically explored neighborhoods and achieved high-quality solutions. Moreover, adaptive perturbation mechanisms combined with efficient local search strategies not only improved solution quality but also enhanced the algorithm's robustness across diverse benchmark instances. These findings highlight the indispensable role of well-designed local search operators in tackling NP-hard problems, striking a crucial balance between exploration and exploitation.

Expanding on the role of local search operators, ref. [21] underscored the advantages of employing parallel search techniques to solve complex combinatorial problems, such as the Set-Union Knapsack Problem (SUKP). The study illustrated how a CUDA-based parallel local search leveraged GPU architectures to optimize the exploration and exploitation processes. Notably, the use of crossover operators, including uniform and two-point, paired with a repair mechanism effectively addressed the infeasible solutions. This innovative integration of parallel processing and local search operators significantly enhanced the scalability and efficiency of the approach, demonstrating its potential to tackle large-scale optimization challenges. These results further emphasized the importance of local search techniques in achieving superior performance across diverse problem domains.

Building on these advancements, recent studies such as [5,22] explored tailored local search strategies to enhance the performance of metaheuristics in solving NP-hard problems. The list-based threshold accepting (LBTA) algorithm introduced in [22] incorporated a dynamic threshold list alongside an improved neighbor operator, achieving a balance between exploration and exploitation. Additionally, a hybrid greedy repair operator was employed to effectively handle infeasible solutions and accelerate convergence. Meanwhile, ref. [5] presented a machine learning-enhanced binary cuckoo search algorithm for the Set-Union Knapsack Problem, featuring a specialized local search operator to refine promising

solutions dynamically. Together, these studies highlight the transformative impact of advanced local search strategies in optimizing performance, improving solution quality, and addressing complex constraints in large-scale optimization scenarios.

3. The Machine Learning Swarm Intelligence Algorithm

This section is dedicated to elaborating on the algorithm designed to tackle the Multi-Demand Multidimensional Knapsack Problem (MDMKP), employing the Whale Optimization Algorithm (WoA) as the core metaheuristic strategy. As depicted in Figure 1, the algorithm's flowchart illustrates the systematic process beginning with solution generation; this is further elaborated in Section 3.1. Following the generation phase, the algorithm assesses whether the termination criteria, defined as the completion of a predetermined number of iterations (set to 800), have been satisfied. The value of 800 iterations was selected based on empirical testing and is detailed in Table 1.

Should the termination criteria remain unmet, the Whale Optimization Algorithm (WoA) would be activated, facilitating the computation of solution velocities. These velocities would then be categorized into clusters, with each cluster being allocated a transition probability that informs the binarization process, as detailed in Section 3.2. The operations conducted by WoA or during binarization might lead to solutions that violate the constraints, necessitating the implementation of a repair operator, which is thoroughly discussed in Section 3.3.

Significantly, this section underscores the evaluation of various configurations of a local search operator to analyze its impact on the algorithm's convergence. The local search operator, integral to refining the newly identified optimal solutions, operates on principles akin to those of the repair operator. By experimenting with different configurations, this study aims to illuminate the local operator's role in enhancing the algorithm's efficiency and convergence towards optimal solutions within the complex landscape of MDMKP.

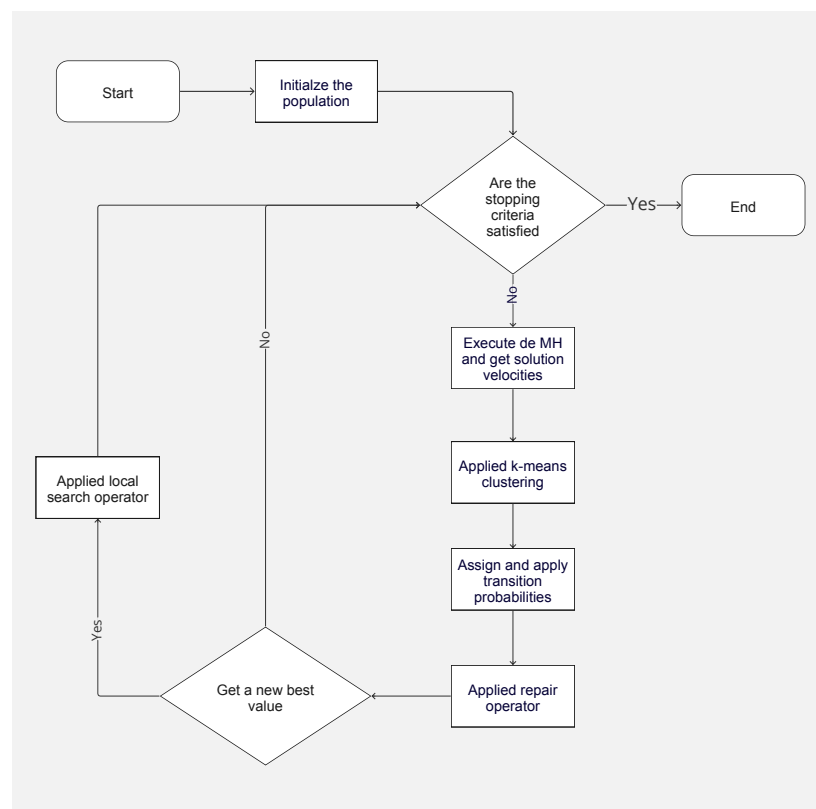


Figure 1. The Developed Swarm Intelligence-Based Machine Learning Algorithm.

Table 1. Setting binarization parameters for WOA.

Parameter	Description	Used Value	Explored Range
P	Particle number	5	[5, 10, 20]
K	Number of groups	5	[4, 5, 6]
I	Swarm iterations	800	[600, 800, 1000, 1200]
L	Local search iterations	2000	[1000, 2000, 4000]
a	Parameter <i>a</i> of WOA: controls exploration and exploitation.	Decreases linearly from 2 to 0	Decreases linearly from 2 to 0
b	Parameter <i>b</i> of WOA: controls spiral shape.	1 (constant)	1 (constant)

3.1. Initialization Operator

The initialization of solutions begins by utilizing the total item count as input variables, denoted as *itemCount*. A random item, represented as *randomItem*, is selected, followed by a verification against the constraints depicted in Equation (2). If compliant, another item is randomly chosen, ensuring that it has not already been included in the solution, denoted by *Solution*. If it is, selection continues until an unlisted item is found, which is then added to *Solution*, and the constraints are re-evaluated. If these are not met, the procedure terminates by removing the last added item, affecting *Solution*. The pseudocode for this process is detailed in Algorithm 1.

Algorithm 1 Solution Initialization Operator

```

1: Function InitializeSolutions(itemCount)
2: Input itemCount
3: Output Solution
4: Solution.append(randint(itemCount))
5: while (knapsack constraints are satisfied) do
6:   randomItem  $\leftarrow$  randint(itemCount)
7:   status  $\leftarrow$  0
8:   while status == 0 do
9:     if randomItem not in Solution then
10:      Solution.append(randomItem)
11:      status  $\leftarrow$  1
12:     end if
13:   end while
14: end while
15: Solution.pop(randomItem)
16: return Solution

```

3.2. ML Binary Operator

The binarization process is driven by the Metaheuristic Binary Transformation (MBT) operator, detailed in the pseudocode of Algorithm 2. This operator refines the solution set, *solSet*, from the previous iteration by applying the chosen metaheuristic strategy (*MetaH*), in this case WoA, in conjunction with the current optimal solution, *optSol*, and the transition probability for each cluster, *clustProb*. The metaheuristic, *MetaH*, updates *solSet* by generating velocities, *velSet*, which represent transition vectors obtained through *MetaH*. These velocities are then processed using k-means clustering, where the number of clusters, *Clusters*, is predefined as 5.

Algorithm 2 Metaheuristic Binary Transformation Operator (MBT)

```

1: Function MBT(solSet, MetaH, clustProb, optSol)
2: Input solSet, MetaH, clustProb
3: Output solSet, optSol
4: velSet  $\leftarrow$  applyMetaHToSolSet(solSet, MetaH) {Applying the metaheuristic}
5: clusteredSol  $\leftarrow$  applyKmeansToVel(velSet, Clusters) {Clustering velocities}
6: for (each soli in clusteredSol) do
7:   for (each dimi,j in soli) do
8:     transProbi,j  $\leftarrow$  assignClustProbToDim(dimi,j, clustProb)
9:     if transProbi,j > randNum then
10:       Update soli,j with the optimal value
11:     else
12:       Retain soli,j as is
13:     end if
14:   end for
15: end for
16: return solSet

```

Each solution, *sol_i*, across its dimensions *d_j*, is assigned to a cluster. The clusters are associated with transition probabilities, *clustProb*, which are ranked according to the magnitude of their centroids. The transition probabilities used are [0.10, 0.20, 0.40, 0.80, 0.90].

The cluster with the smallest centroid is assigned the lowest transition probability, 0.10. Lower centroid magnitudes correspond to smaller transition probabilities, *clustProb*. For each dimension *d_{i,j}* of *sol_i*, a specific transition probability *transProb_{i,j}* is computed and compared with a randomly generated number, *randNum*. If *transProb_{i,j}* exceeds *randNum*, *sol_{i,j}* is updated to the best observed value; otherwise, it remains unchanged.

3.3. Repair Operator and Local Search Operator

This section aims to explain the repair algorithm, which is also used to identify a new optimum as part of the local search process. Algorithm 3 presents the pseudocode for the repair and local search operator. The operator starts by taking the solution (*Sol*) obtained from the binarized metaheuristic as input and initializes two variables—*candidates* and *isImproved*. The algorithm continues while *isImproved* remains as 0. A movement—either add, drop, swap, or randomly one of the three—is selected to perform the repair and local search for improvement. The impact of each operator is analyzed to compare their effects. If the add operation is selected, elements not currently in the solution are sequentially added and evaluated using the penalty function defined in Equations (4) and (5), as proposed by [12].

$$P(Sol) = \sum_{i=1}^m \max \left(0, \sum_{j=1}^n a_{ij}x_j - c_i \right) + \sum_{i=q+1}^{q+m} \max \left(0, c_i - \sum_{j=1}^n a_{ij}x_j \right) \quad (4)$$

$$F(Sol) = \sum_{j=1}^n p_j x_j - \lambda \times P(Sol) \quad (5)$$

Solutions that do not meet the constraints can still be incorporated if they improve the penalized objective function *F*, defined in (5). This process is repeated for each element not currently in the solution. If the addition of an element improves the penalized objective function over the previous one, the element is added to the solution. In Equation (4), the penalty applied depends on whether $\sum_{j=1}^n$ exceeds or falls short of *c_i*, with the penalty decreasing as $\sum_{j=1}^n$ approaches *c_i*. The same procedure is applied for dropping an item or swapping elements between the solution and those not in it. Finally, the candidates are

checked to determine if any of them satisfy the constraints, and if so, the best candidate is selected and *isImproved* is updated to 1. In the case of the local search operator, the same movements are used, but the algorithm runs for a fixed number of iterations (2000). *Sol* is a valid solution and, at the end of the iterations, it is verified whether a better valid solution was found than the original one.

Algorithm 3 Enhanced Repair and Local Search Operator

```

1: Function enhancedRepairAndLocalSearch(Sol, operationType, iterationLimit)
2: Input Sol, operationType, iterationLimit
3: Output Sol
4: candidates  $\leftarrow$  []
5: isImproved  $\leftarrow$  FALSE
6: iteration  $\leftarrow$  0
7: while iteration < iterationLimit do
8:   if operationType == "add" then
9:     newSol  $\leftarrow$  addItem(Sol)
10:  else if operationType == "drop" then
11:    newSol  $\leftarrow$  dropItem(Sol)
12:  else if operationType == "swap" then
13:    newSol  $\leftarrow$  swapItem(Sol)
14:  else if operationType == "mix" then
15:    randomOperation  $\leftarrow$  randomly choose between "add", "drop", or "swap"
16:    if randomOperation == "add" then
17:      newSol  $\leftarrow$  addItem(Sol)
18:    else if randomOperation == "drop" then
19:      newSol  $\leftarrow$  dropItem(Sol)
20:    else if randomOperation == "swap" then
21:      newSol  $\leftarrow$  swapItem(Sol)
22:    end if
23:  end if
24:  if newSol improves based on Equation (5) then
25:    candidates.append(newSol)
26:    isImproved  $\leftarrow$  TRUE
27:    Sol  $\leftarrow$  newSol
28:  end if
29:  iteration  $\leftarrow$  iteration + 1
30: end while
31: if isImproved then
32:   Sol  $\leftarrow$  selectBestCandidate(candidates)
33: end if
34: return Sol

```

Algorithm 3 provides a flexible framework to study the impact of different repair strategies in optimization problems. The following four configurations are explored: Addition only, Drop only, Swap only, and a mixed strategy incorporating all three. Each configuration helps analyze the effects of the corresponding operation on solution improvement, as follows:

- **Addition Only:** This strategy focuses solely on adding new elements to *Sol*, testing whether incremental additions can improve the penalized objective function while maintaining constraints.

- **Drop Only:** This approach evaluates the effect of dropping elements from *Sol* to streamline the solution, analyzing if reducing complexity enhances performance.
- **Swap Only:** The “Swap” operation involves exchanging elements between *Sol* and those outside of it to explore whether rearranging elements can optimize the solution.
- **Mixed Operations:** This dynamic approach applies a random combination of “Addition”, “Drop”, and “Swap” operations, aiming to identify whether a hybrid strategy leads to a more efficient solution repair.

The algorithm begins by invoking the ‘repairSolution’ function, which takes the current solution *Sol* and the selected experiment type (*operationType*: ‘add’, ‘drop’, ‘swap’, or ‘mix’) as input. The operator generates and evaluates candidate solutions (*candidates*) based on their contribution to the improvements in the penalized objective function and constraint satisfaction. If at least one candidate satisfies the constraints, the best candidate is selected as the new optimal solution.

This method offers valuable insights into the design of more effective metaheuristic algorithms by systematically examining the role of repair and local search operations in driving convergence towards optimal solutions.

3.4. Whale Optimization Algorithm (WOA)

The Whale Optimization Algorithm (WOA) is a bio-inspired metaheuristic based on the hunting behavior of humpback whales, particularly their use of “bubble-net feeding”. Proposed by Mirjalili and Lewis in [11], WOA mimics the whales’ strategies for searching, encircling, and attacking prey. These behaviors are mathematically modeled to solve optimization problems.

The algorithm initializes with a randomly generated population of solutions (whales). During each iteration, the position of each whale is updated based on either the best solution found so far or a randomly chosen whale. This ensures a balance between exploration and exploitation of the search space.

1. **Identifying the Prey.** The prey in WOA represents the current best solution. Whales adjust their position towards this solution using coefficient vectors $\vec{A}$ and $\vec{C}$, which are defined as follows:

$$\vec{A} = 2 \cdot \vec{a} \cdot \vec{r} - \vec{a}, \quad \vec{C} = 2 \cdot \vec{r}$$

where $\vec{a}$ decreases linearly from 2 to 0 over the iterations, while $\vec{r}$ is a random vector in $[0, 1]$. These coefficients enable WOA to dynamically shift between the exploration ($|\vec{A}| \geq 1$) and exploitation ($|\vec{A}| < 1$) phases.

2. **Encircling the Prey.** Once the prey is identified, whales move closer, mimicking the encircling behavior. This is modeled as follows:

$$\vec{X}_i^{t+1} = \vec{X}^* - \vec{A} \cdot |\vec{C} \cdot \vec{X}^* - \vec{X}_i^t|$$

where $\vec{X}^*$ represents the best solution so far, and $\vec{X}_i^t$ is the position of the i -th whale at iteration t .

3. **Bubble-Net Attacking Technique.** WOA simulates the spiral motion and shrinking circle of bubble-net feeding. The position update combines spiral motion and encircling with a 50% probability:

$$\vec{X}_i^{t+1} = \begin{cases} \vec{X}^* - \vec{A} \cdot |\vec{C} \cdot \vec{X}^* - \vec{X}_i^t|, & \text{if } p < 0.5 \\ |\vec{X}^* - \vec{X}_i^t| \cdot e^{b \cdot l} \cdot \cos(2\pi l) + \vec{X}^*, & \text{if } p \geq 0.5 \end{cases}$$

where b defines the spiral shape, l is a random number in $[-1, 1]$, and p is a random probability.

4. Results

This section describes the local search operator experiments conducted on MDMKP instances. The approach for parameter tuning is detailed in Section 4.1, and the comparison of various local search strategies is outlined in Section 4.2. Each instance was executed 30 times, and the best, average, and worst outcomes were recorded for all the experiments. The results are presented in tables and illustrated through boxplot charts. The implementation was carried out in Python 3.8, with performance optimizations enabled by the numba library [23].

This study assessed the performance of algorithms using three sets of reference instances from <https://people.brunel.ac.uk/~mastjjb/jeb/info.html>, (accessed on 20 December 1994).

The first set comprised 48 small instances with $n = 100$, the second set had 48 medium-sized instances with $n = 250$, and the third set included 30 large instances with $n = 500$. For the first two sets, the number of knapsack constraints m ranged from 5 to 10, and the number of demand constraints q varied between 2 and 10. In the 500-instance set, both m and q were fixed at 30.

The naming convention for the problem instances followed the format $n-m-q-i-j$. Here, n represents the number of items, m denotes the number of knapsack constraints, and q specifies the number of demand constraints. The indices i and j correspond to variations in the parameters and random seeds used to generate the instances. For example, the instance name “100-5-2-0-0” refers to an instance with 100 items, five knapsack constraints, and two demand constraints, where the first two indices i and j are set to 0 for parameter variation.

4.1. Parameter Setting for WoA

The parameter configuration was conducted following the method described in [24]. This approach utilized four metrics, as defined in Equations (6)–(9). For parameter determination, the instances 100-5-x-x-x from Table 2 were selected. The results are presented in Table 1, focusing solely on the adjustment of WOA parameters.

The binary WOA requires tuning specific parameters, including the number of particles (P), the number of groups (K) when using the k-means technique, the number of iterations (I) performed by the algorithm, and the number of iterations executed by the local search operator (L). These parameters were adjusted using the percentile binarization technique.

The four metrics used for evaluation are described as follows:

1. The deviation of the best value obtained from the best-known value:

$$bSolution = 1 - \frac{KnownBestValue - BestValue}{KnownBestValue} \quad (6)$$

2. The deviation of the worst value obtained from the best-known value:

$$wSol = 1 - \frac{KnownBestValue - WorstValue}{KnownBestValue} \quad (7)$$

3. The deviation of the average value obtained from the best-known value:

$$aSol = 1 - \frac{KnownBestValue - AverageValue}{KnownBestValue} \quad (8)$$

4. The normalized convergence time during execution:

$$nTime = 1 - \frac{convergenceTime - minTime}{maxTime - minTime} \quad (9)$$

Table 2. Performance comparison for different local operator configurations on 100 instances using the Whale Optimization Algorithm. The table summarizes the maximum and average values, average computation time, and standard deviation for each configuration.

Instance	Best Known	Max_Value	Add Configuration			Max_Value	All Configuration			Max_Value	Drop Configuration			Max_Value	Swap Configuration		
			Avg_Value	Avg_Time	Std_Time		Avg_Value	Avg_Time	Std_Time		Avg_Value	Avg_Time	Std_Time		Avg_Value	Avg_Time	Std_Time
100-5-2-0-0	28,384	27,400	25,528.7	0.23	0.16	28,384	28,131.7	0.19	0.08	28,384	28,146.37	0.39	0.22	27,400	25,456.5	0.24	0.16
100-5-2-0-1	26,386	25,688	24,030.07	0.26	0.16	26,386	26,241.57	0.23	0.09	26,345	26,184.17	0.36	0.21	25,665	24,005.33	0.28	0.20
100-5-2-0-2	23,484	22,898	21,298.7	0.30	0.15	23,484	23,332.67	0.21	0.08	23,484	23,372.27	0.54	0.15	22,898	21,002.83	0.27	0.18
100-5-2-0-3	27,374	25,884	24,407.27	0.25	0.16	27,298	27,276.5	0.16	0.07	27,298	27,276.57	0.34	0.16	25,931	24,299.9	0.27	0.19
100-5-2-0-4	30,632	28,878	27,034.6	0.27	0.16	30,632	30,555.4	0.19	0.09	30,632	30,555.6	0.46	0.20	29,703	27,015.87	0.29	0.19
100-5-2-0-5	44,674	44,225	42,974.73	0.32	0.21	44,614	44,473.77	0.27	0.08	44,614	44,490.43	0.63	0.21	43,930	42,690.27	0.34	0.23
100-5-2-1-0	10,379	10,312	10,114.37	0.21	0.18	10,364	10,243.13	0.19	0.09	10,312	10,235.6	0.44	0.21	10,287	10,061.33	0.27	0.19
100-5-2-1-1	11,114	11,066	10,427.4	0.21	0.15	11,114	11,089.9	0.18	0.09	11,114	11,094.37	0.36	0.18	11,067	10,319.83	0.25	0.21
100-5-2-1-2	10,124	9835.4	9835.4	0.26	0.17	9993	9940.93	0.17	0.09	9993	9948.27	0.36	0.16	9983	9791.43	0.27	0.19
100-5-2-1-3	10,567	10,421	10,041.73	0.27	0.20	10,567	10,545.8	0.17	0.09	10,567	10,528.33	0.43	0.17	10,550	10,034.53	0.32	0.18
100-5-2-1-4	10,658	10,522	10,110.1	0.16	0.14	10,522	10,485.8	0.17	0.08	10,522	10,477.57	0.32	0.19	10,412	10,042.73	0.15	0.14
100-5-2-1-5	17,550	17,504	17,268.57	0.27	0.22	17,537	17,483.57	0.21	0.11	17,545	17,482.13	0.45	0.25	17,498	17,343.47	0.34	0.24
100-5-5-0-0	21,892	21,892	21,249.9	0.31	0.23	21,892	21,790.9	0.13	0.11	21,892	21,749.3	0.46	0.25	21,892	21,028.77	0.32	0.24
100-5-5-0-1	26,280	26,036	25,005.53	0.21	0.17	26,280	26,278	0.13	0.09	26,280	26,278.33	0.39	0.25	26,036	24,981.6	0.34	0.24
100-5-5-0-2	20,628	20,362	19,925.27	0.33	0.21	20,628	20,628	0.13	0.03	20,628	20,628	0.33	0.10	20,580	20,036.07	0.40	0.22
100-5-5-0-3	21,547	21,204	20,605.33	0.29	0.20	21,526	21,454.7	0.24	0.11	21,526	21,468.9	0.53	0.26	21,367	20,673.37	0.25	0.20
100-5-5-0-4	25,074	24,862	24,285.6	0.37	0.21	25,067	25,064.87	0.23	0.09	25,067	25,061.7	0.48	0.24	25,066	24,487.07	0.45	0.23
100-5-5-0-5	40,327	40,256	39,972.1	0.48	0.31	40,299	40,194.33	0.26	0.14	40,243	40,186.23	0.60	0.30	40,246	39,936.17	0.46	0.33
100-5-5-1-0	10,263	10,210	10,091.57	0.25	0.14	10,263	10,151.07	0.21	0.11	10,263	10,155.77	0.42	0.28	10,210	10,055.63	0.27	0.18
100-5-5-1-1	10,625	10,463	10,138.23	0.13	0.18	10,625	10,579	0.20	0.10	10,625	10,589.8	0.39	0.23	10,604	10,099.13	0.23	0.23
100-5-5-1-2	10,198	10,126	9815.23	0.23	0.20	10,126	10,028.77	0.18	0.13	10,126	10,018.5	0.45	0.24	10,126	9852.6	0.13	0.15
100-5-5-1-3	10,030	9864	9674.03	0.23	0.16	9998	9895.7	0.18	0.09	9998	9884.37	0.52	0.23	9998	9658.5	0.24	0.22
100-5-5-1-4	9964	9785	9639.5	0.11	0.12	9838	9773.9	0.18	0.12	9802	9770.87	0.38	0.23	9808	9692.97	0.13	0.18
100-5-5-1-5	15,603	15,603	15,430.77	0.24	0.29	15,591	15,548.53	0.26	0.12	15,591	15,555.53	0.50	0.27	15,591	15,455.03	0.21	0.20
100-10-5-0-0	21,852	21,520	20,950.07	0.51	0.29	21,715	21,668.47	0.23	0.14	21,715	21,667.83	0.51	0.36	21,649	20,931.67	0.42	0.30
100-10-5-0-1	20,645	20,397	19,788.63	0.36	0.27	20,593	20,357.23	0.27	0.12	20,593	20,363.83	0.64	0.31	20,360	19,764.77	0.36	0.26
100-10-5-0-2	19,517	19,296	18,919.6	0.40	0.25	19,517	19,296.03	0.20	0.12	19,517	19,287.53	0.66	0.33	19,434	18,943.5	0.30	0.25
100-10-5-0-3	20,596	20,452	19,831.43	0.33	0.26	20,485	20,363.93	0.21	0.12	20,454	20,350.43	0.48	0.31	20,356	19,696.3	0.35	0.26
100-10-5-0-4	19,423	19,041	18,654.13	0.45	0.32	19,264	19,036.53	0.25	0.14	19,155	19,035.57	0.67	0.35	19,155	18,800.4	0.34	0.27
100-10-5-0-5	35,933	35,539	35,062.53	0.51	0.33	35,816	35,551.9	0.37	0.13	35,631	35,486.63	0.71	0.44	35,534	35,038.1	0.46	0.34
100-10-5-1-0	10,018	10,018	9720.5	0.21	0.27	9896	9767.87	0.18	0.12	10,018	9785.07	0.51	0.30	9880	9691.3	0.25	0.20
100-10-5-1-1	9839	9690	9570.27	0.28	0.21	9690	9658.37	0.20	0.12	9690	9662.6	0.42	0.32	9690	9480.33	0.38	0.20
100-10-5-1-2	10,000	9498	9434.97	0.36	0.27	9672	9545	0.26	0.14	9672	9532.4	0.54	0.30	9600	9461.93	0.35	0.26
100-10-5-1-3	10,544	10,430	10,243.43	0.26	0.23	10,544	10,397.27	0.24	0.14	10,510	10,391.23	0.62	0.32	10,430	10,202.97	0.18	0.19
100-10-5-1-4	10,011	9875	9678.33	0.17	0.20	9904	9809	0.26	0.13	9904	9828.63	0.58	0.27	9838	9643.73	0.21	0.19
100-10-5-1-5	16,230	16,095	15,861.53	0.39	0.35	16,101	15,931.6	0.28	0.17	16,174	15,954.7	0.89	0.40	16,150	15,824.03	0.30	0.28
100-10-10-0-0	22,054	21,849	21,471.8	0.52	0.28	22,054	21,868.53	0.22	0.14	22,054	21,871.27	0.49	0.36	21,849	21,370.8	0.46	0.27
100-10-10-0-1	20,103	20,103	19,872.33	0.32	0.33	20,103	20,077.47	0.27	0.16	20,103	20,077	0.61	0.31	20,103	19,814.43	0.35	0.29
100-10-10-0-2	19,381	19,381	19,057.73	0.51	0.38	19,229	19,119.37	0.31	0.15	19,268	19,106.77	0.61	0.34	19,208	18,995.2	0.36	0.27
100-10-10-0-3	17,434	17,372	17,142.17	0.19	0.24	17,372	17,329.43	0.26	0.17	17,372	17,313.13	0.64	0.35	17,416	17,148.4	0.22	0.27
100-10-10-0-4	18,792	18,769	18,514.4	0.30	0.30	18,769	18,637.57	0.26	0.17	18,769	18,650.47	0.53	0.34	18,769	18,553.87	0.27	0.23
100-10-10-0-5	33,837	33,595	33,162.17	0.54	0.43	33,736	33,631.8	0.39	0.19	33,736	33,630.83	0.88	0.48	33,631	33,289.47	0.53	0.34
100-10-10-1-0	8560	8560	8398.7	0.48	0.31	8451	8328.57	0.30	0.14	8475	8373.6	0.51	0.32	8501	8360.63	0.44	0.33
100-10-10-1-1	8493	8493	8413.87	0.20	0.21	8493	8457	0.20	0.15	8493	8467.8	0.53	0.33	8493	8443.8	0.13	0.09
100-10-10-1-2	9266	9158	9140.97	0.30	0.35	9227	9180.57	0.23	0.13	9227	9189.77	0.55	0.37	9227	9149.47	0.29	0.36
100-10-10-1-3	9823	9823	9770.2	0.36	0.26	9823	9652.77	0.29	0.16	9823	9630.5	0.54	0.37	9823	9748.1	0.27	0.18
100-10-10-1-4	8929	8929	8890.43	0.40	0.27	8929	8861.6	0.24	0.14	8929	8856.33	0.69	0.42	8929	8902.77	0.33	0.27
100-10-10-1-5	14,152	14,099	14,061.27	0.52	0.38	14,151	14,082.57	0.34	0.22	14,099	14,061.6	0.84	0.54	14,106	14,032.53	0.75	0.44
Average	18,108.10	17,864.02	17,385.75	0.31	0.24	18,052.52	17,954.15	0.23	0.12	18,046.50	17,952.39	0.52	0.29	17,895.40	17,360.61	0.31	0.23
p-value						1.65×10^{-17}	2.61×10^{-6}										
Kruskal Wallis																	

Note: The bold values represent the best results obtained in terms of maximum value (Max_Value) and average value (Avg_Value) for each configuration.

4.2. Comparative Analysis of Local Search Operators

This section focuses on comparing the local search operator across different configurations to assess its performance and impact on solving the Multi-Demand Multidimensional Knapsack Problem. The local search operator, described in Section 3.3, is evaluated under the following four distinct configurations: Addition only, Drop only, Swap only, and a mixed strategy combining all three.

To facilitate the comparison, each instance was executed 30 times, where the *max_value* represented the best value obtained across these 30 runs, and the *avg_max_value* corresponded to the average of the maximum values achieved in these 30 executions. Summary tables are constructed for the four types of instances studied, detailing key metrics such as the maximum value achieved (*max_value*), the average of the maximum values (*avg_value*), the average computational time (*avg_time*), and the standard deviation of the computational time (*std_time*) for each configuration across all instances.

Furthermore, violin plots are used to visually represent the distribution of results, providing insights into the variability and concentration of the obtained values. Meanwhile, the scatter plots compare the %GAP of average and maximum values against the best-known solutions, where %GAP stands for “percentage gap”, defined as $\%GAP = \left(\frac{\text{Best Known Value} - \text{Obtained Value}}{\text{Best Known Value}} \right) \times 100\%$, quantifying the relative difference between the obtained solution and the best-known solution. These plots illustrate how each configuration performs as the instance size or solution quality changes. To rigorously assess the statistical significance of the observed differences among the configurations, we employed the Kruskal–Wallis test [25], a non-parametric method suitable for comparing multiple groups without assuming a normal distribution. Whenever significant differences arose, the post hoc Dunn’s test [26] would be used to pinpoint the specific operator pairs that exhibited statistically significant differences.

4.2.1. Performance Analysis on Small-Sized Instances ($n = 100$)

The results for small-sized instances ($n = 100$) are presented in this section, focusing on the performance of four local operator configurations within the Whale Optimization Algorithm. Table 2 shows the maximum and average values obtained for each configuration, along with the average computation times and standard deviations. Statistical significance among the configurations was evaluated using the Kruskal–Wallis test, confirming the significant differences ($p < 0.05$) in both the average and maximum results. Notably, a higher *max_value* indicates that a configuration is capable of finding superior solutions under optimal conditions, while a lower standard deviation in computation time reflects a more consistent performance across multiple runs. These characteristics are particularly important for the knapsack problem, as they suggest that configurations with high *max_value* and low variability can deliver more reliable and predictable outcomes in practical applications.

Pairwise comparisons between configurations are detailed in Table 3, where the post hoc Dunn’s test was applied. This analysis identifies statistically significant differences between specific operator pairs. For example, the significant p -values observed between the “Add” and “All Operator” configurations suggest that integrating multiple operators enhances the algorithm’s capacity to both explore new regions of the solution space or exploit promising areas of the current solution. Similarly, the significant differences between the “All Operator” and “Drop” configurations indicate that relying solely on removal operations limits the search dynamics, whereas a combined approach fosters a more balanced and effective search strategy. These results highlight the superior performance of the “All Operator” configuration. In contrast, no significant differences were observed between the “Add” and “Swap” configurations, suggesting similar behavior in these instances.

Figure 2 provides a visual summary of the %GAP results for different local operator configurations applied to the instances with $n = 100$. The violin plots in the top row illustrate the distribution and variability of %GAP values for averages and maximums across the “Add”, “Swap”, “All Operator”, and “Drop” configurations. The violin plots reveal that the “All Operator” configuration not only achieves a higher median performance but also exhibits a narrower spread, indicating more consistent results across runs. This consistency is further corroborated by the scatter plots, which show lower variability in the %GAP values relative to the best-known solution for the “All Operator” configuration. These observations underscore the robustness and reliability of the “All Operator” strategy in achieving superior performance.

The scatter plots in the bottom row compare the %GAP values for the averages (left panel) and maximums (right panel) relative to the best-known solution. These plots reveal trends in the operator performance across different problem instances, showing that the “All Operator” configuration has both superior performance and lower variability compared to the other configurations.

Table 3. Post hoc Dunn’s test results for pairwise comparisons between the operators ($n = 100$). Avg p -value and Max p -value indicate significance levels, with corresponding “Yes” or “No” for significant differences.

Comparison	Avg p -Value	Max p -Value	Avg Sig.	Max Sig.
Add vs. All Operator	1.56×10^{-9}	3.88×10^{-4}	Yes	Yes
Add vs. Drop	1.00	1.00	No	No
Add vs. Swap	1.78×10^{-9}	6.29×10^{-4}	Yes	Yes
All Operator vs. Drop	6.90×10^{-10}	1.40×10^{-3}	Yes	Yes
All Operator vs. Swap	1.43×10^{-6}	1.65×10^{-5}	Yes	Yes
Drop vs. Swap	7.89×10^{-10}	2.20×10^{-3}	Yes	Yes

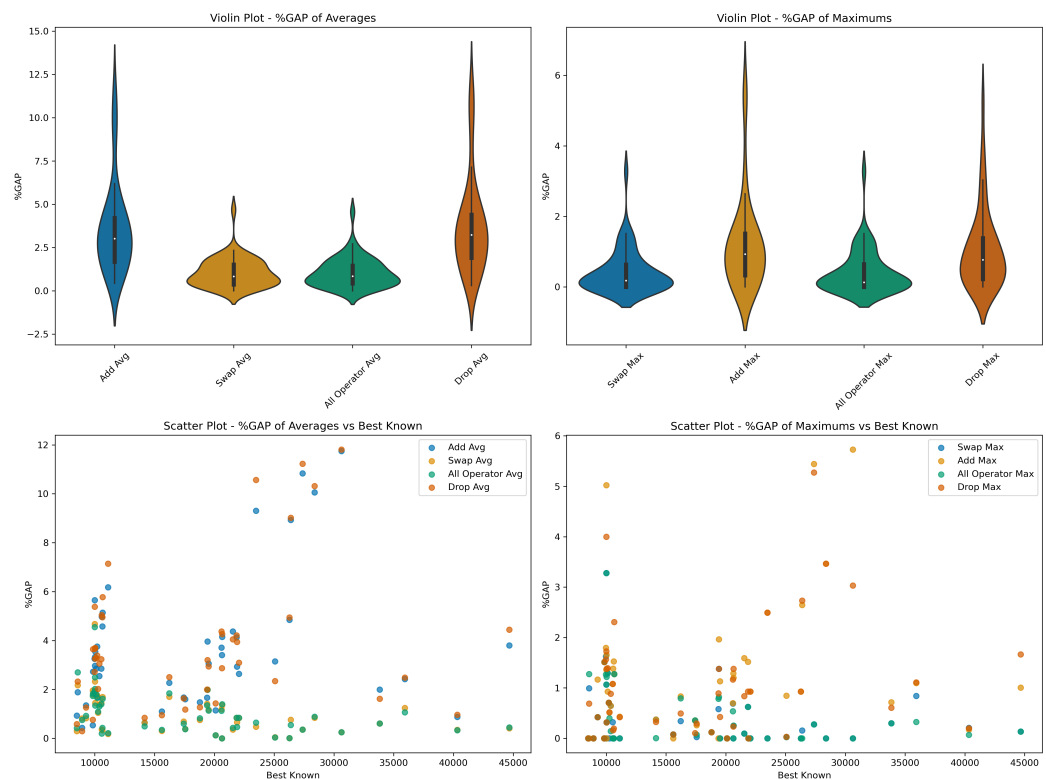


Figure 2. Violin plots (top row) show the distribution of %GAP values for averages and maximums across configurations. Scatter plots (bottom row) compare %GAP values relative to the best-known solution for averages (left) and maximums (right) for small-sized instances ($n = 100$).

4.2.2. Performance Analysis on Medium-Sized Instances ($n = 250$)

The results for medium-sized instances ($n = 250$) are presented in this section, highlighting the performance of four local operator configurations within the Whale Optimization Algorithm framework. Table 4 details the maximum and average values obtained across configurations, along with the associated computation times and standard deviations. Statistical significance among configurations was assessed using the Kruskal–Wallis test, with p -values confirming significant differences ($p < 0.05$) in both the average and maximum results. Pairwise comparisons, shown in Table 5, were performed using the post-hoc Dunn’s test, which identified the statistically significant differences between specific operator pairs for both the average and maximum values. Finally, Figure 3 visually summarizes the performance across configurations. Scatter plots contrast the %GAP of average and maximum values relative to the best-known solution, while violin plots emphasize the distribution and variability of %GAP values. These visualizations and analyses underline the effectiveness and variability of each configuration, particularly the superior performance of the “All Operator” configuration in many cases.

The results presented in Table 4 provide a detailed comparison of the four local operator configurations applied to medium-sized instances ($n = 250$) within the Whale Optimization Algorithm framework. The table highlights the maximum and average values achieved by each configuration, as well as the associated computation times and standard deviations. Notably, the “All Operator” configuration consistently outperforms the other configurations in terms of both the maximum and average values, with statistically significant differences confirmed by the Kruskal–Wallis test ($p < 0.05$). These differences can be attributed to the synergistic effect of combining multiple local operators, which enables a more balanced search strategy that both explores diverse regions of the solution space and exploits promising areas more effectively. Although this integrated approach may slightly increase computation time, the resultant improvement in solution quality is a favorable trade-off. In practical applications, particularly for medium-sized instances of the Multi-Demand Multidimensional Knapsack Problem, this robust performance translates into more reliable and consistent outcomes, thereby enhancing the overall effectiveness of the optimization process. These findings suggest that the “All Operator” configuration offers a more robust exploration of the solution space. This result underscores the effectiveness of the “All Operator” configuration in optimizing performance for this instance size and algorithmic setup.

The results in Table 5 summarize the pairwise comparisons of values between the four local operator configurations using the post hoc Dunn’s test. This analysis complements the overall significance identified by the Kruskal–Wallis test, providing detailed insights into which specific operator pairs exhibit statistically significant differences. The “Add” configuration shows significant differences with the “All Operator” configuration in both the average and maximum values ($p < 0.05$), underscoring the superior performance of the latter. Similarly, significant differences are observed between the “All Operator” and “Drop” configurations, as well as the “All Operator” and “Swap” configurations. These findings reaffirm the effectiveness of the “All Operator” configuration, particularly when compared to the other approaches. In contrast, the “Add” and “Swap” configurations show no significant differences in performance, suggesting similar behaviors in the tested instances.

Table 4. Performance comparison for different local operator configurations on 250 instances using the Whale Optimization Algorithm. The table presents pairwise comparisons of results, with statistical significance assessed through the Kruskal–Wallis test.

Instance	Best Known	Max_Value	Add Configuration			Max_Value	All Configuration			Max_Value	Drop Configuration			Max_Value	Swap Configuration		
			Avg_Value	Avg_Time	Std_Time		Avg_Value	Avg_Time	Std_Time		Avg_Value	Avg_Time	Std_Time		Avg_Value	Avg_Time	Std_Time
250-5-2-0-0	78,486	73,206	68,716.67	1.47	1.04	77,933	77,576.97	1.45	0.32	73,261	68,892.60	1.72	0.94	77,400	76,514.30	0.46	0.08
250-5-2-0-1	75,132	71,099	66,969.60	1.61	1.04	74,532	73,912.00	1.57	0.27	71,872	67,537.43	1.65	0.99	74,054	72,573.13	0.47	0.05
250-5-2-0-2	71,003	66,213	63,701.03	1.94	1.09	70,270	69,470.73	1.61	0.20	66,192	63,921.67	2.10	0.90	69,262	68,027.07	0.45	0.09
250-5-2-0-3	80,311	76,363	71,820.67	2.10	1.08	80,038	79,865.40	1.32	0.34	74,828	71,509.80	1.72	1.00	79,780	79,455.43	0.46	0.07
250-5-2-0-4	70,935	66,767	63,235.17	1.71	1.09	70,630	70,490.33	1.32	0.40	66,801	63,576.60	1.82	0.81	70,378	70,098.77	0.50	0.06
250-5-2-0-5	130,981	126,876	123,875.40	2.39	1.28	128,620	127,653.67	1.93	0.29	126,842	123,941.50	2.49	1.38	127,784	126,115.07	0.54	0.14
250-5-2-1-0	26,666	25,971	25,304.63	1.83	1.07	26,330	26,224.43	1.37	0.37	26,208	25,514.70	1.81	1.15	26,219	25,971.67	0.42	0.09
250-5-2-1-1	26,864	26,557	25,727.30	1.62	0.95	26,599	26,491.67	1.14	0.38	26,482	25,545.27	1.71	0.98	26,584	26,392.77	0.41	0.10
250-5-2-1-2	27,280	26,513	25,745.33	1.74	1.04	26,988	26,875.90	1.27	0.35	26,760	25,764.80	1.62	0.94	26,836	26,672.77	0.46	0.09
250-5-2-1-3	26,269	26,049	25,655.97	1.52	1.04	25,958	25,840.73	1.23	0.36	25,997	25,560.63	1.57	0.92	25,910	25,692.30	0.38	0.14
250-5-2-1-4	27,293	26,902	26,107.80	2.14	0.95	27,043	26,892.37	1.20	0.38	26,880	26,033.37	1.73	0.96	26,971	26,673.10	0.45	0.07
250-5-2-1-5	44,395	44,311	43,862.70	2.33	1.42	44,185	43,986.33	1.43	0.59	44,185	43,818.27	2.74	1.29	44,159	43,925.20	0.37	0.17
250-5-5-0-0	68,026	65,526	63,403.77	2.49	1.25	67,954	67,792.20	1.47	0.43	66,238	63,265.10	1.90	1.18	67,802	67,404.80	0.54	0.09
250-5-5-0-1	60,795	59,706	58,105.03	2.50	1.27	60,461	60,203.73	1.69	0.46	59,714	58,207.43	2.70	1.29	60,126	59,750.53	0.50	0.16
250-5-5-0-2	62,093	60,353	58,026.33	2.41	1.32	62,054	61,888.17	1.52	0.46	60,360	58,628.37	2.57	1.27	61,939	61,620.17	0.54	0.09
250-5-5-0-3	66,567	64,824	63,305.23	2.47	1.10	66,375	66,264.50	1.57	0.39	64,743	62,990.33	2.32	1.30	66,143	65,819.10	0.56	0.09
250-5-5-0-4	61,929	60,440	58,217.30	2.58	1.42	61,883	61,772.10	1.19	0.56	59,473	58,015.83	2.61	1.21	61,798	61,691.30	0.49	0.12
250-5-5-0-5	127,934	126,504	124,516.70	2.71	1.57	127,284	126,822.33	2.35	0.45	126,501	124,353.17	3.31	1.71	126,959	125,871.47	0.70	0.13
250-5-5-1-0	26,973	26,659	26,492.43	1.73	1.26	26,896	26,562.07	1.17	0.70	26,712	26,434.00	1.82	1.26	26,630	26,403.93	0.38	0.17
250-5-5-1-1	26,665	26,349	26,198.97	2.51	1.26	26,368	26,181.50	1.16	0.49	26,433	26,179.30	2.12	1.34	26,245	26,037.33	0.36	0.15
250-5-5-1-2	26,648	26,349	26,104.53	2.12	1.34	26,212	26,087.57	1.16	0.61	26,272	26,068.07	2.23	1.29	26,198	25,980.33	0.43	0.16
250-5-5-1-3	25,923	25,521	25,387.43	1.87	1.20	25,522	25,359.13	1.07	0.60	25,575	25,394.13	1.75	1.32	25,408	25,261.70	0.40	0.17
250-5-5-1-4	26,060	25,610	25,421.73	1.81	1.46	25,594	25,490.13	1.20	0.67	25,783	25,469.73	1.44	1.17	25,662	25,405.83	0.37	0.15
250-10-5-0-0	56,306	54,579	53,549.43	2.66	1.60	55,588	55,145.17	1.89	0.66	55,004	53,866.30	2.87	1.62	55,176	54,627.00	0.60	0.19
250-10-5-0-1	59,619	58,279	55,446.03	2.19	1.24	59,298	59,100.10	2.09	0.62	57,612	55,705.53	2.78	1.59	59,048	58,655.40	0.66	0.16
250-10-5-0-2	54,898	53,217	52,178.43	3.18	1.27	54,284	53,898.07	2.05	0.66	53,331	51,876.57	2.64	1.69	54,152	53,322.23	0.58	0.19
250-10-5-0-3	52,399	51,195	50,660.67	2.93	1.53	51,776	51,438.17	2.06	0.71	51,478	50,511.53	3.02	1.41	51,375	51,140.13	0.59	0.17
250-10-5-0-4	58,234	56,774	55,484.47	2.79	1.30	57,425	56,871.20	1.94	0.65	56,610	55,244.67	2.59	1.49	57,102	56,167.73	0.59	0.19
250-10-5-0-5	99,682	98,690	97,822.50	3.99	1.77	98,872	98,442.10	2.09	1.12	98,921	97,825.60	3.55	1.97	98,938	98,119.50	0.72	0.24
250-10-5-1-0	26,961	26,532	26,149.27	3.91	1.14	26,632	26,319.60	1.60	0.70	26,436	26,131.00	2.84	1.48	26,428	26,177.37	0.47	0.19
250-10-5-1-1	26,658	26,288	25,902.53	2.49	1.55	26,256	26,136.77	1.58	0.77	26,412	25,974.27	2.56	1.70	26,243	25,985.13	0.51	0.19
250-10-5-1-2	25,737	25,348	25,102.90	2.27	1.38	25,222	25,073.83	1.22	0.75	25,274	25,086.80	3.25	1.47	25,203	25,004.13	0.38	0.18
250-10-5-1-3	27,162	26,849	26,471.57	3.04	1.40	26,809	26,619.30	1.69	0.67	26,772	26,507.83	3.17	1.53	26,882	26,509.60	0.44	0.19
250-10-5-1-4	26,816	26,364	26,001.37	2.27	1.28	26,434	26,277.37	1.30	0.70	26,431	25,987.13	2.34	1.68	26,304	26,125.90	0.45	0.20
250-10-5-1-5	46,244	45,821	45,519.73	3.34	2.00	45,952	45,754.67	1.79	0.88	45,899	45,632.23	2.59	1.79	45,835	45,581.37	0.59	0.21
250-10-10-0-0	52,441	51,927	51,288.50	3.17	1.92	52,048	51,809.77	2.20	0.83	51,760	51,329.83	3.35	1.91	51,839	51,526.40	0.69	0.21
250-10-10-0-1	53,745	53,222	52,726.53	2.85	2.17	53,502	53,329.97	2.14	0.68	53,338	52,700.53	3.66	1.75	53,394	53,162.07	0.66	0.24
250-10-10-0-2	46,927	46,382	46,066.27	3.36	1.57	46,516	46,286.03	2.14	0.73	46,525	46,022.67	3.27	1.64	46,374	46,087.30	0.64	0.25
250-10-10-0-3	54,831	54,412	53,549.03	3.41	1.68	54,571	54,346.03	2.25	0.77	54,382	53,589.97	3.36	2.24	54,534	54,049.87	0.59	0.24
250-10-10-0-4	49,675	49,363	48,816.07	3.09	2.15	49,379	49,175.40	2.07	1.01	49,418	48,734.23	3.06	1.83	49,234	48,951.83	0.59	0.23
250-10-10-0-5	92,975	92,414	91,758.60	4.94	2.55	92,609	92,246.10	2.58	1.23	92,573	91,772.97	4.47	2.24	92,304	91,883.13	0.71	0.35
250-10-10-1-0	26,696	26,389	26,266.57	2.41	2.06	26,448	26,281.13	1.10	0.71	26,502	26,273.33	3.24	1.72	26,292	26,136.03	0.55	0.26
250-10-10-1-1	25,876	25,577	25,367.40	2.69	1.84	25,548	25,328.00	1.53	0.98	25,628	25,342.47	2.63	1.64	25,396	25,223.70	0.56	0.28
250-10-10-1-2	26,517	26,306	25,995.67	1.96	1.71	26,236	25,964.33	1.52	0.86	26,098	25,952.47	3.20	2.06	26,016	25,822.43	0.53	0.28
250-10-10-1-3	26,684	26,167	26,004.70	2.88	1.69	26,295	25,993.83	1.83	0.90	26,221	26,014.60	3.06	1.83	26,079	25,827.17	0.52	0.28
250-10-10-1-4	26,676	26,431	26,121.80	2.21	1.62	26,359	26,105.73	1.71	0.84	26,344	26,119.17	3.09	2.06	26,143	26,019.17	0.51	0.21
250-10-10-1-5	42,629	42,056	41,874.47	3.50	2.58	42,066	41,830.17	2.03	1.05	42,038	41,862.33	4.30	2.08	41,925	41,682.47	0.84	0.28
average	49,853.90	48,670.46	47,557.83	2.55	1.46	49,431.19	49,170.08	1.64	0.63	48,666.10	47,571.11	2.62	1.47	49,235.69	48,826.33	0.52	0.17
p-value						9.87×10^{-6}	5.69×10^{-12}										
Kruskal Wallis																	

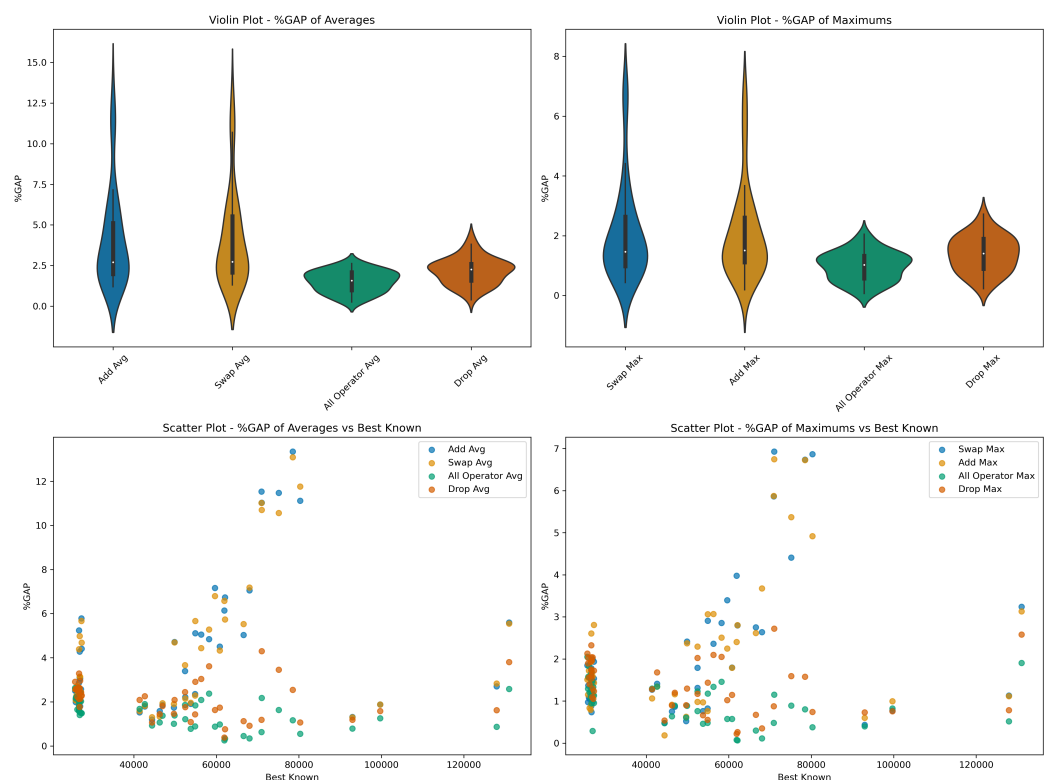
Note: The bold values represent the best results obtained in terms of maximum value (Max_Value) and average value (Avg_Value) for each configuration.

Table 5. Post hoc Dunn’s test results for pairwise comparisons between operators. Avg p -value and Max p -value indicate significance levels, with corresponding “Yes” or “No” for significant differences.

Comparison	Avg p -Value	Max p -Value	Avg Sig.	Max Sig.
Add vs. All Operator	1.99×10^{-9}	5.10×10^{-5}	Yes	Yes
Add vs. Drop	1.15×10^{-2}	6.71×10^{-1}	Yes	No
Add vs. Swap	1.00	1.00	No	No
All Operator vs. Drop	8.82×10^{-3}	2.53×10^{-2}	Yes	Yes
All Operator vs. Swap	8.68×10^{-10}	7.60×10^{-5}	Yes	Yes
Drop vs. Swap	7.42×10^{-3}	7.97×10^{-1}	Yes	No

Figure 3 presents a visualization of the %GAP results for different local operator configurations applied to 250 instances using the Whale Optimization Algorithm. The violin plots in the top row illustrate the distribution and variability of %GAP values for both the averages and maximums across the “Add”, “Swap”, “All Operator”, and “Drop” configurations. These plots highlight the concentration of values and the spread of performance for each operator, with the “All Operator” configuration showing the most consistent and favorable results.

The scatter plots in the bottom row provide a comparative perspective, contrasting the %GAP values for averages (left panel) and maximums (right panel) relative to the best-known solution. These plots reveal trends in operator performance across varying problem instances, emphasizing the superior performance and lower variability of the “All Operator” configuration compared to the others. Overall, the figure offers both statistical and visual evidence of the effectiveness and reliability of each configuration.

**Figure 3.** Comparison of %GAP across different operators for average and maximum values relative to the best-known solution. The left panel shows the scatter plots of %GAP for averages, while the right panel displays %GAP for maximums. The violin and scatter plots below illustrate the distribution of %GAP for averages and maximums, emphasizing the variability and performance of each operator.

4.2.3. Performance Analysis on Large-Sized Instances ($n = 500$)

The results for larger instances ($n = 500$) are presented in this section, focusing on the performance of four local operator configurations within the Whale Optimization Algorithm. As shown in Table 6, each configuration's maximum and average values are reported alongside computation times and their standard deviations. A Kruskal–Wallis test ($p < 0.05$) confirmed that these configurations produce statistically different outcomes in both the average and maximum values. Subsequently, Table 7 provides post hoc Dunn's test comparisons, illustrating which specific operator pairs exhibit significant differences. Collectively, these analyses offer a comprehensive view of how each configuration behaves on instances of size $n = 500$.

Within Table 6, the “All Operator” configuration again demonstrates notably strong results, often surpassing the other three configurations in terms of both maximum and average performance. For certain instances, the “Add” or “Swap” configurations yield competitive outcomes, but their results typically carry greater variability. In contrast, the “Drop” configuration tends to produce considerably lower average values, which is further highlighted by the large gap in some instances. This performance gap and increased variability observed in the “Drop” configuration may be due to its limited ability to recover from suboptimal moves, as it solely removes items without providing opportunities for corrective actions such as adding or swapping. Consequently, this approach might miss promising regions in the solution space, leading to less consistent outcomes. Additionally, the superior performance of the “All Operator” configuration suggests that combining diverse local moves fosters a more balanced search strategy, which is particularly beneficial as problem size increases, thereby enhancing the algorithm's scalability and robustness. These findings reinforce the idea that simultaneously leveraging multiple local operators (“All Operator”) can better explore the solution space, leading to improved overall outcomes at this instance scale.

The post hoc Dunn's test results shown in Table 7 substantiate the overall Kruskal–Wallis findings by pinpointing exactly where the significant differences occur. Notably, “Drop” diverges substantially from the other configurations in most of the pairwise comparisons (e.g., “Add vs. Drop”, “All Operator vs. Drop”, and “Swap vs. Drop”). Meanwhile, both “Add vs. All Operator” and “Add vs. Swap” comparisons show strong statistical significance, supporting the notion that “All Operator” remains a top performer across multiple measures. In contrast, certain pairs (like “Drop vs. Swap”) occasionally demonstrate no significant difference in specific tests, suggesting that those methods can converge to similar results under particular conditions, albeit at generally lower performance levels than “All Operator”.

A visual summary of these outcomes for $n = 500$ can be provided in a figure analogous to Figure 3, now adapted for larger instances (e.g., Figure 4). The violin plots would illustrate the distribution and spread of %GAP in both the averages and maximums for each operator, while the scatter plots would relate %GAP to the best-known values. These visualizations highlight the pronounced difference in performance among configurations and underscore the relatively consistent advantage of “All Operator”. By combining the statistical and graphical insights, it becomes clear that the richer local search provided by the “All Operator” configuration often yields superior solutions and reduced variability for $n = 500$.

Table 6. Performance comparison for different local operator configurations on 500 instances using the Whale Optimization Algorithm. The best *max_value* and *avg_value* in each row (for this maximization problem) are highlighted in bold.

Instance	Best Known	Add Configuration				All Configuration				Drop Configuration				Swap Configuration			
		Max_Value	Avg_Value	Avg_Time	Std_Time	Max_Value	Avg_Value	Avg_Time	Std_Time	Max_Value	Avg_Value	Avg_Time	Std_Time	Max_Value	Avg_Value	Avg_Time	Std_Time
500-30-30-0-2-1	85,188	84,250.00	83,663.10	15.65	6.64	84,452.00	83,917.50	15.35	6.75	79,123.00	24,250.27	11.48	17.68	84,134.00	83,558.77	17.32	1.83
500-30-30-0-2-2	82,073	81,281.00	80,611.43	12.84	6.65	81,312.00	80,788.90	15.15	7.17	79,652.00	35,848.40	14.25	17.81	81,033.00	80,493.20	19.24	1.87
500-30-30-0-2-3	77,393	76,113.00	75,520.53	15.21	6.47	75,993.00	75,681.93	12.25	6.93	70,793.00	14,783.07	5.65	12.35	76,435.00	75,452.10	17.24	1.87
500-30-30-0-2-4	82,304	81,293.00	80,860.40	13.86	6.06	81,380.00	80,932.73	14.91	8.41	77,007.00	33,244.70	14.90	18.05	81,327.00	80,674.37	18.48	2.00
500-30-30-0-2-5	83,525	82,662.00	81,931.03	12.57	6.00	82,541.00	82,089.37	15.07	7.05	75,422.00	23,519.07	10.23	17.19	82,319.00	81,870.43	17.36	1.78
500-30-30-0-2-6	145,967	144,754.00	144,364.03	17.40	9.06	144,679.00	144,321.60	17.46	8.46	137,567.00	60,253.30	14.68	20.51	144,745.00	144,162.80	27.04	2.54
500-30-30-0-2-7	152,246	151,077.00	150,755.67	22.11	9.46	151,221.00	150,756.83	14.14	8.72	147,797.00	100,777.50	22.69	22.22	151,104.00	150,538.87	22.48	1.98
500-30-30-0-2-8	157,687	156,725.00	156,120.20	14.28	8.36	156,860.00	156,206.00	17.68	8.74	155,735.00	91,456.97	22.93	23.06	156,510.00	155,870.23	29.20	1.93
500-30-30-0-2-9	153,751	152,543.00	152,220.43	19.44	10.06	152,684.00	152,204.57	22.93	8.37	148,989.00	100,922.67	26.13	24.18	152,397.00	152,029.17	25.36	2.27
500-30-30-0-2-10	142,173	141,013.00	140,641.17	18.00	9.58	141,031.00	140,682.97	19.59	9.92	137,382.00	76,161.57	21.88	22.65	140,868.00	140,392.00	24.52	2.22
500-30-30-0-2-11	185,226	184,069.00	183,773.73	16.27	8.02	184,266.00	183,800.73	13.94	7.90	177,110.00	97,618.50	16.71	18.20	183,982.00	183,589.33	18.48	1.79
500-30-30-0-2-12	194,614	193,521.00	193,173.73	11.94	5.86	193,464.00	193,165.80	11.17	6.33	186,230.00	48,899.70	7.90	14.51	193,441.00	192,986.07	19.32	1.81
500-30-30-0-2-13	208,246	207,399.00	207,073.57	13.53	5.85	207,318.00	207,065.67	13.67	6.64	203,997.00	91,981.10	11.32	15.51	207,293.00	206,935.73	19.04	1.50
500-30-30-0-2-14	215,849	214,861.00	214,428.97	13.45	7.06	214,825.00	214,531.67	13.39	6.97	210,392.00	95,518.33	14.21	17.89	214,805.00	214,331.30	15.00	1.83
500-30-30-0-2-15	194,224	193,152.00	192,812.63	15.29	7.11	193,389.00	192,876.60	14.62	7.11	188,763.00	79,229.17	11.83	16.41	193,235.00	192,657.17	18.52	1.47
500-30-30-1-5-1	51,666	50,224.00	49,821.30	15.02	5.55	50,694.00	49,903.53	15.14	6.64	40,515.00	11,264.50	12.88	18.76	50,199.00	49,509.03	16.72	1.79
500-30-30-1-5-2	50,101	48,549.00	48,148.41	19.39	33.01	48,748.00	48,182.50	15.76	7.60	33,490.00	10,654.47	11.02	15.20	48,284.00	47,877.83	18.76	1.75
500-30-30-1-5-3	51,226	31,640.00	30,921.10	14.43	19.75	49,614.00	48,942.97	14.53	6.87	31,640.00	8,070.50	12.12	18.94	49,388.00	48,671.83	19.00	1.67
500-30-30-1-5-4	51,637	34,043.00	33,988.33	32.65	13.65	50,454.00	50,061.27	12.91	7.61	34,043.00	18,443.27	29.95	11.21	50,117.00	49,692.10	18.20	1.66
500-30-30-1-5-5	52,078	43,466.00	42,323.10	13.23	18.90	51,126.00	50,460.40	14.32	7.60	43,466.00	12,703.70	11.88	17.70	50,740.00	50,241.27	19.96	1.57
500-30-30-1-5-6	84,052	82,489.00	81,980.30	27.54	9.21	82,858.00	82,382.13	19.28	8.22	77,550.00	28,525.40	16.05	19.69	82,489.00	81,981.20	24.52	2.42
500-30-30-1-5-7	82,850	81,299.00	80,693.32	28.65	12.43	81,438.00	81,063.77	19.36	7.54	67,723.00	28,979.63	20.92	23.67	81,299.00	80,700.43	25.04	2.16
500-30-30-1-5-8	82,722	81,400.00	80,734.20	22.65	7.54	81,613.00	81,107.77	18.51	9.57	71,409.00	37,218.90	25.80	22.95	81,400.00	80,736.90	23.92	1.90
500-30-30-1-5-9	82,825	81,289.00	80,381.50	28.11	9.97	81,220.00	80,685.33	31.52	6.78	72,527.00	46,741.80	33.74	21.97	81,289.00	80,384.00	27.92	2.04
500-30-30-1-5-10	82,845	81,299.00	80,828.43	15.29	36.03	81,388.00	80,752.33	22.08	9.02	64,891.00	40,560.90	32.78	26.06	80,985.00	80,366.03	26.64	2.07
500-30-30-1-5-11	88,887	87,861.00	87,376.73	14.59	6.17	87,632.00	87,331.10	13.94	5.78	68,347.00	21,356.87	12.86	18.87	87,621.00	86,976.57	18.56	1.31
500-30-30-1-5-12	87,254	86,305.00	85,792.37	15.42	5.78	86,356.00	85,855.40	17.33	5.08	74,836.00	16,042.73	7.69	14.51	85,932.00	85,434.27	19.48	1.58
500-30-30-1-5-13	87,315	86,520.00	86,071.40	14.84	6.68	86,659.00	86,082.67	14.72	5.23	79,544.00	41,096.90	16.35	17.29	86,488.00	85,901.73	18.04	1.67
500-30-30-1-5-14	87,583	86,527.00	86,205.87	16.23	5.94	86,594.00	86,165.73	14.59	6.93	69,624.00	39,298.07	17.87	15.97	86,346.00	85,890.07	18.16	1.43
500-30-30-1-5-15	87,956	87,085.00	86,769.03	15.32	6.78	87,204.00	86,801.17	15.78	6.03	69,359.00	27,718.33	11.15	16.17	86,914.00	86,491.93	17.84	1.73
average	109,049	106,490.30	105,999.52	17.51	10.32	107,967.10	107,493.36	16.37	7.40	99,164.10	45,438.01	16.66	18.57	107,770.63	107,213.22	20.71	1.85
p-value						1.52×10^{-14}	1.48×10^{-11}										
Kruskal Wallis																	

Table 7. Post hoc Dunn’s test results for pairwise comparisons between operators ($n = 500$). Avg p -value and Max p -value indicate significance levels, with corresponding “Yes” or “No” for significant differences.

Comparison	Avg p -Value	Max p -Value	Avg Sig.	Max Sig.
Add vs. All Operator	1.43×10^{-4}	1.14×10^{-4}	Yes	Yes
Add vs. Drop	1.39×10^{-10}	6.56×10^{-8}	Yes	Yes
Add vs. Swap	3.05×10^{-9}	1.57×10^{-7}	Yes	Yes
All Operator vs. Drop	7.70×10^{-12}	5.99×10^{-10}	Yes	Yes
All Operator vs. Swap	2.45×10^{-9}	3.21×10^{-8}	Yes	Yes
Drop vs. Swap	1.00	1.00	No	No

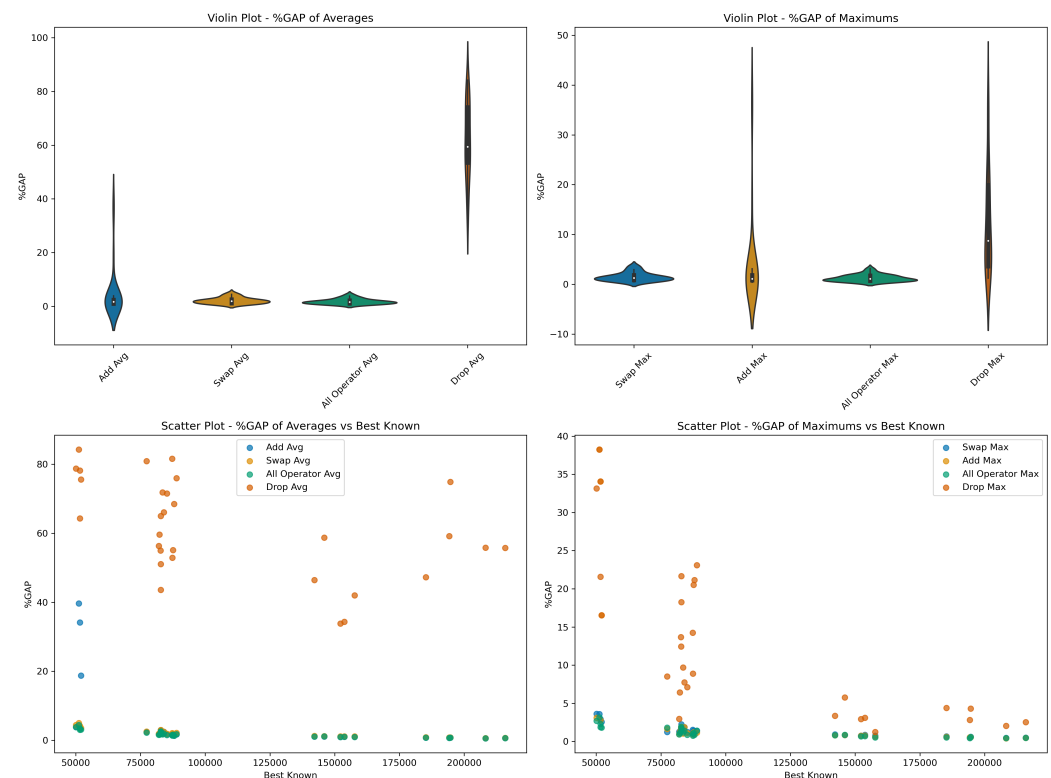


Figure 4. Comparison of %GAP across different operators for $n = 500$ when evaluating average and maximum values relative to the best-known solution. The left panel presents scatter plots of %GAP for average results, and the right panel shows %GAP for maximum outcomes. Below these, the violin and scatter plots illustrate the distribution of %GAP, emphasizing the performance and variability among the operators.

In conclusion, for these larger problem sizes, the “All Operator” configuration demonstrates a robust capability to produce higher-quality solutions compared to the other operators. The statistical tests confirm that many of these advantages are significant, with the post hoc Dunn’s test further detailing the operator pairs responsible for the performance gaps. Despite some competitive instances for “Add” or “Swap”, the overall evidence supports the conclusion that combining multiple local operators typically leads to the better exploitation and exploration of the solution space, making it a compelling strategy for $n = 500$ instances in the Whale Optimization Algorithm.

To validate the results using more than one algorithm, the Grey Wolf Optimizer (GWO) [27] was employed to solve the challenging MDMKP problem with a population size of $n = 30$. The metaheuristic optimization algorithm, inspired by the social behavior and hunting strategies of grey wolves, follows a leadership hierarchy consisting of the

following four types of wolves: alpha, beta, delta, and omega. The first three guide the search process, which is structured in three main phases as follows: searching for prey, encircling the prey, and attacking the prey. To assess its effectiveness in addressing large-scale instances of the MDMKP problem, the algorithm was tested on large-scale instances under different local operator configurations.

The number of search agents (*SearchAgents_no*) was set to five, defining the population size, while the maximum number of iterations (*Max_iter*) was established at 800, determining the number of updates in the agents' positions. The control parameter a was utilized to balance exploration and exploitation and was defined as $a = 2 - \frac{2 \times l}{Max_iter}$, where l represents the current iteration number, starting at $a = 2$. Two coefficients, A and C , were used to adjust the wolves' movements during the optimization. The coefficient A was calculated as $A = 2 \times a \times r_1 - a$, where r_1 is a random number in the range $[0, 1]$, allowing A to vary between $[-2, 2]$. Meanwhile, C was computed as $C = 2 \times r_2$, where r_2 is another random number in $[0, 1]$, ensuring that C remains within the range $[0, 2]$.

The main results from Table 8 indicate that the "All Operator" consistently achieved the best performance across most instances. In terms of the *max_value* and *avg_value*, the "All Operator" outperformed the other local operators in nearly all cases. Furthermore, the "All Operator" demonstrated competitive average execution times and standard deviations, indicating a stable and efficient performance. From the overall comparison, it can be concluded that the "All Operator" provided the most robust results in terms of solution quality and computational efficiency. The Kruskal–Wallis test confirmed statistically significant differences ($p\text{-value} < 0.05$), reinforcing the superiority of this operator over the other tested alternatives.

Table 8. Performance comparison for different local operator configurations on 500 instances using the GWO Algorithm. The best *max_value* and *avg_value* in each row (for this maximization problem) are highlighted in bold.

Instance	Add Configuration				All Configuration				Drop Configuration				Swap Configuration			
	Max_Value	Avg_Value	Avg_Time	Std_Time	Max_Value	Avg_Value	Avg_Time	Std_Time	Max_Value	Avg_Value	Avg_Time	Std_Time	Max_Value	Avg_Value	Avg_Time	Std_Time
500-30-30-0-2-1	78,318	75,867.90	12.07	2.31	84,242	83,819.00	13.97	7.62	84,122	83,487.83	15.82	2.63	78,233	75,425.83	17.45	3.57
500-30-30-0-2-2	74,878	72,653.57	14.13	3.22	81,212	80,752.23	16.75	6.85	80,890	80,382.67	16.14	2.33	75,427	72,727.23	18.32	4.12
500-30-30-0-2-3	69,724	67,519.07	17.07	2.41	75,934	75,648.60	14.42	7.86	75,964	75,332.47	15.67	2.69	70,293	67,363.23	15.67	2.95
500-30-30-0-2-4	75,590	72,877.17	18.54	3.10	81,393	80,962.30	12.57	7.37	81,364	80,725.00	16.02	2.69	75,244	72,981.57	16.78	2.67
500-30-30-0-2-5	77,163	73,602.43	15.24	4.05	82,700	82,113.13	14.31	7.07	82,459	81,799.33	16.60	2.54	77,336	73,862.20	19.56	3.45
500-30-30-0-2-6	137,966	134,213.23	20.11	2.72	144,708	144,339.37	20.84	8.92	144,761	144,239.70	14.39	7.98	137,853	134,695.53	22.00	4.50
500-30-30-0-2-7	144,574	141,802.37	16.33	2.91	151,466	150,876.27	15.86	9.81	151,169	150,560.33	19.34	3.95	144,980	141,870.27	17.89	3.67
500-30-30-0-2-8	150,026	146,484.40	19.77	2.54	156,513	156,203.07	22.63	8.01	156,449	155,910.40	18.09	3.15	148,975	146,813.07	15.45	3.80
500-30-30-0-2-9	145,260	141,896.20	18.22	2.41	152,536	152,250.37	18.37	9.00	152,901	152,119.97	17.95	4.01	145,910	142,122.07	21.34	3.50
500-30-30-0-2-10	135,548	131,049.40	16.44	3.21	141,030	140,783.80	18.89	8.30	141,111	140,474.30	18.49	3.54	134,112	131,221.17	17.77	2.88
500-30-30-0-2-11	178,091	175,851.97	19.01	4.68	184,096	183,817.17	12.56	5.65	184,175	183,695.87	15.35	2.27	178,516	176,070.10	20.55	3.33
500-30-30-0-2-12	187,629	184,714.03	15.99	2.84	193,583	193,232.37	14.40	7.21	193,466	193,074.60	16.71	2.24	187,100	184,716.63	16.88	3.95
500-30-30-0-2-13	200,929	198,293.60	17.66	3.67	207,407	207,036.47	14.27	7.40	207,305	206,941.47	16.10	2.75	200,840	198,681.80	15.12	2.55
500-30-30-0-2-14	208,063	204,997.57	21.45	4.12	215,047	214,612.87	14.49	6.47	214,692	214,349.37	15.67	2.64	207,939	205,310.27	16.22	3.87
500-30-30-0-2-15	187,493	184,711.40	20.55	3.33	193,290	192,859.70	12.65	7.16	193,149	192,660.73	15.89	2.31	187,956	184,965.63	15.89	3.44
500-30-30-1-5-1	43,978	40,155.43	16.12	2.65	50,314	49,859.77	14.14	7.54	50,241	49,724.13	15.20	2.77	42,837	39,326.87	17.65	4.05
500-30-30-1-5-2	42,297	39,319.50	15.76	3.21	48,693	48,182.13	13.05	7.19	48,587	47,956.27	16.78	2.29	41,886	39,108.57	18.99	3.88
500-30-30-1-5-3	42,833	39,210.97	22.78	2.77	49,472	48,953.67	15.50	7.16	49,740	48,853.30	16.45	2.59	41,448	39,037.63	19.88	4.50
500-30-30-1-5-4	46,755	42,082.20	15.34	3.55	50,375	50,010.67	15.27	7.48	50,418	49,841.33	15.49	2.36	44,620	41,229.83	18.21	2.41
500-30-30-1-5-5	44,664	41,296.03	16.58	3.12	50,854	50,437.43	15.10	6.57	50,698	50,300.67	16.14	2.87	44,529	41,379.40	17.43	4.68
500-30-30-1-5-6	74,719	72,120.07	20.00	4.22	82,704	82,339.57	20.88	9.45	82,485	82,027.97	17.99	3.24	75,226	72,251.97	21.67	3.44
500-30-30-1-5-7	73,698	70,226.27	18.45	2.91	81,576	81,102.30	19.55	9.52	81,422	80,887.00	17.17	3.20	73,771	70,326.70	15.67	2.85
500-30-30-1-5-8	74,474	70,188.50	18.88	3.68	81,443	81,054.90	19.66	9.15	81,562	80,866.00	18.69	3.04	72,377	69,645.63	21.22	4.50
500-30-30-1-5-9	73,777	69,578.83	17.55	2.99	81,226	80,765.57	20.76	8.52	80,956	80,380.03	18.94	3.42	72,794	69,043.47	15.34	3.75
500-30-30-1-5-10	70,880	68,177.33	16.78	3.47	81,266	80,733.00	22.69	9.83	81,006	80,453.37	18.75	2.55	72,804	68,362.60	20.45	2.91
500-30-30-1-5-11	80,417	77,042.10	18.90	4.68	87,680	87,330.23	18.30	7.32	87,593	87,095.43	16.19	2.60	80,719	77,506.07	21.01	3.33
500-30-30-1-5-12	79,851	75,944.27	19.55	2.84	86,408	85,840.10	15.82	6.72	86,104	85,486.93	16.28	2.57	78,572	75,923.63	17.66	3.77
500-30-30-1-5-13	80,032	76,123.40	20.23	3.95	86,458	86,109.60	14.61	6.64	86,340	85,916.50	16.10	2.43	78,903	76,439.60	16.54	4.10
500-30-30-1-5-14	80,523	77,083.70	17.88	2.77	86,705	86,211.73	15.13	5.55	86,549	85,946.10	16.42	2.21	79,089	76,610.50	18.66	3.59
500-30-30-1-5-15	80,571	77,038.67	15.99	3.33	87,092	86,739.00	15.24	7.06	87,100	86,552.17	16.35	2.73	80,941	77,595.03	16.45	4.45
average	101,357.37	98,070.72	17.78	3.26	107,914.10	107,499.21	16.42	7.68	107,825.93	107,268.04	16.71	2.95	101,041.00	98,087.14	18.12	3.62
p-value Kruskal Wallis					0.0064	0.0051										

4.3. Computational Complexity Analysis

The computational complexity of the proposed algorithm can be divided into the following two main components: (a) the metaheuristic process (Whale Optimization Algorithm, binarization, and repair) and (b) the local search operator, which is executed conditionally when a new global optimum is found.

The metaheuristic process, applied at every iteration, involves several key steps. First, the metaheuristic updates the positions and velocities of the population, which incurs a complexity of $O(N \times M)$, where N represents the population size and M is the number of items in the problem. Subsequently, k -means clustering is applied to the velocity vectors, introducing a cost of $O(\ell \times N \times k \times M)$, where ℓ is the number of iterations required for k -means convergence and k is the number of clusters. Additional operations, such as binarization and solution repair, also exhibit a complexity of $O(N \times M)$. Combining these steps, the overall complexity for the metaheuristic process over T iterations can be expressed as follows:

$$O\left(T \times (N \times M + \ell \times N \times k \times M)\right).$$

This formulation keeps the metaheuristic component generic, allowing for flexibility in substituting the Whale Optimization Algorithm with other metaheuristic approaches without altering the overall structure of the analysis.

The local search operator, outlined in Algorithm 3, is executed only when a new global optimum is found. During its execution, it iteratively applies one of the predefined movements—addition, removal, swap, or a combination of these (mix)—up to a specified iteration limit, *iterationLimit*. Each movement involves examining items in the solution or candidate set. The complexity depends on the type of movement selected, as follows

- For *addItem* or *dropItem*, the worst-case complexity is $O(M)$ per iteration, as these movements involve checking the inclusion or exclusion of items and evaluating the penalized objective function.
- For *swapItem*, the worst-case complexity increases to $O(M^2)$, as this operator evaluates potential exchanges between the items inside and outside the solution.
- For *mix*, which combines all movements probabilistically, the complexity is dominated by $O(M^2)$ due to the inclusion of *swapItem*.

Given that the local search runs for at most *iterationLimit* iterations per execution, the complexity of a single invocation is as follows:

$$O(\text{iterationLimit} \times M) \quad (\text{for addItem or dropItem}).$$

$$O(\text{iterationLimit} \times M^2) \quad (\text{for swapItem or mix}).$$

The local search operator is not executed during every iteration but only when the algorithm identifies a new global optimum. In practice, the number of such activations (α) is significantly smaller than the total number of iterations (T), as the frequency of finding a new global optimum decreases over time. Empirically, this behavior often follows a geometric or exponential decay pattern.

In the worst-case scenario, where a new optimum is found in every iteration ($\alpha = T$), the total cost of the local search would be the following:

$$O(T \times \text{iterationLimit} \times M) \quad (\text{for addItem or dropItem}).$$

$$O(T \times \text{iterationLimit} \times M^2) \quad (\text{for swapItem or mix}).$$

In realistic scenarios, where α represents only a fraction of T (e.g., $\alpha = 0.1 T$), the contribution of the local search to the overall complexity is substantially reduced and can be approximated as follows:

$$O(\alpha \times iterationLimit \times M) \quad (\text{for addItem or dropItem}).$$

$$O(\alpha \times iterationLimit \times M^2) \quad (\text{for swapItem or mix}).$$

Combining the complexity of the metaheuristic process and the conditionally executed local search, the overall complexity of the algorithm can be approximated as follows:

$$O\left(T \times (N \times M + \ell \times N \times k \times M)\right) + O(\alpha \times iterationLimit \times M) \quad (\text{for addItem or dropItem}).$$

$$O\left(T \times (N \times M + \ell \times N \times k \times M)\right) + O(\alpha \times iterationLimit \times M^2) \quad (\text{for swapItem or mix}).$$

4.4. Sensitivity Analysis for Local Search Iterations (L)

The sensitivity analysis focuses on the parameter L , representing the number of local search iterations, as outlined in Table 1. The objective is to evaluate the performance of different iteration counts ($L = 1000, 1200, 1400, 1600, 1800, 2000$) for large instances ($n = 500$) of the MDMKP. For each iteration count, ten executions were performed for every instance, and two metrics were computed, with the maximum value (max_value) representing the best value obtained across the executions, and the average maximum value (avg_max_value) representing the mean of the best values from each execution.

To standardize comparisons across instances, the percentage gap (%GAP) was calculated for both metrics. Unlike comparisons against a known optimal or best-known value, the %GAP in this experiment measured the deviation of the obtained value from the best value across all iterations for each metric. The %GAP is calculated as follows:

$$\%GAP = \left(\frac{\text{Best Value} - \text{Obtained Value}}{\text{Best Value}} \right) \times 100$$

In Figure 5, the first graph (Subfigure (a)) illustrates the %GAP of the maximum value (max_value_gap) across the iterations. Each line represents a specific instance, showcasing how the maximum performance deviates from the best value obtained across all iterations as the number of iterations increases.

The second graph (Subfigure (b)) depicts the %GAP of the average maximum value (avg_max_value_gap) over the iterations. Similar to the first graph, each line corresponds to a unique instance. This chart provides insight into the consistency of average performance as the number of iterations varies.

The results presented in Figure 5 indicate that increasing the number of local search iterations (L) generally leads to a reduction in the percentage gap (%GAP) for both metrics—the maximum value (max_value) and the average maximum value (avg_max_value). In Subfigure (a), it can be observed that the %GAP of the maximum value decreases consistently as L increases, with the lowest %GAP values achieved at $L = 2000$. A similar trend is observed in Subfigure (b), where the %GAP of the average maximum value also decreases steadily, reaching its minimum at $L = 2000$. This behavior is consistent across all instances analyzed.

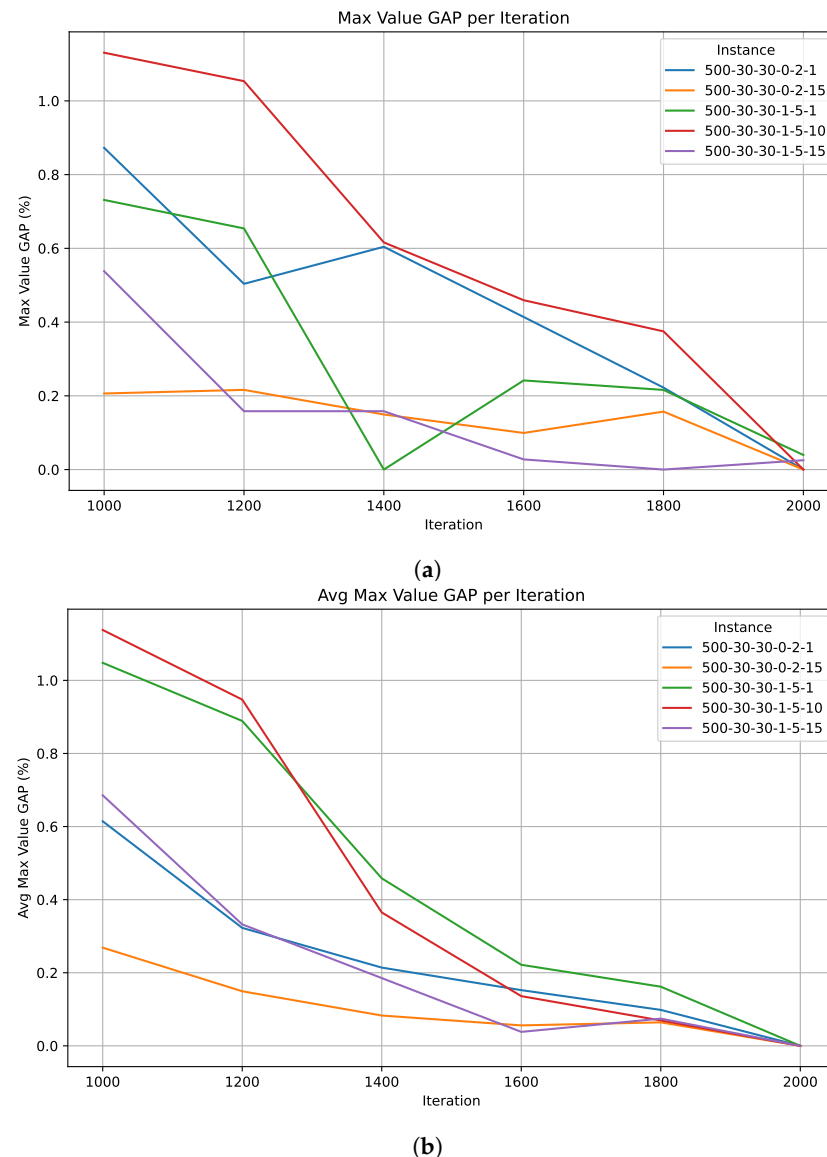


Figure 5. Sensitivity analysis results for the parameter L . (a) %GAP of maximum value. (b) %GAP of average maximum value.

It is interesting to note that slight fluctuations in %GAP were observed around $L \approx 1200$ – 1400 for some instances. However, the overall trend continued decreasing, with the most significant reductions in %GAP observed as L approached 2000. These results indicate that while the intermediate values of L may yield temporary variations, higher values consistently provide better outcomes.

In conclusion, $L = 2000$ is recommended as the optimal parameter for local search iterations, as it consistently produces superior results with reduced variability across all analyzed instances. The stability and improved performance at $L = 2000$ highlight its effectiveness in producing solutions closer to the best-obtained values for all instances.

5. Conclusions

In this study, the impact of four local operator configurations—Add, Drop, Swap, and All Operator—on the performance of metaheuristic algorithms in solving the MDMKP was investigated. The experiments were conducted using the Whale Optimization Algorithm (WoA) with a specialized binarization process and a repair mechanism. Each local operator configuration was evaluated on small ($n = 100$), medium ($n = 250$), and large ($n = 500$) benchmark instances. The results consistently indicated that the use of multiple local oper-

ators in the “All Operator” configuration led to significantly better performance compared to individual operators. Higher average and maximum solution values were achieved, while maintaining greater stability. In contrast, the “Drop Operator” was found to be less effective in large-scale settings. Although “Add” and “Swap” performed competitively on certain instances, their results exhibited greater variability. Statistical analyses, including the Kruskal–Wallis and Dunn’s post hoc tests, confirmed that these differences were statistically significant in most pairwise comparisons.

To further validate these findings, an additional study was conducted using the Grey Wolf Optimizer (GWO) under the same four local operator configurations. The results again indicated that the “All Operator” configuration yielded the best performance for large-scale instances, reinforcing the conclusion that incorporating multiple local moves improves both solution quality and consistency. Additionally, a sensitivity analysis was performed on the local search iteration parameter (L), varying its value from 1000 to 2000 in increments. The results showed a consistent reduction in the percentage gap (%GAP) for both the maximum and average maximum values as L increased. The most reliable improvements were observed at $L = 2000$, where the lowest %GAP values were consistently recorded across all instances.

Several directions for future research can be considered. The development of adaptive or learning-based approaches that dynamically adjust operator usage based on the search state could enhance the balance between exploration and exploitation, particularly in large-scale or real-time optimization contexts. Additionally, extending the “All Operator” strategy to other combinatorial optimization problems (e.g., knapsack variants, scheduling, or routing) could provide further insights into its broader applicability and robustness. Investigating parallelization strategies may also be a promising avenue, especially for large-scale problems, given the computational complexity of advanced local operators combined with sophisticated metaheuristics.

Author Contributions: Conceptualization, J.G. and P.M.; Methodology, I.C. and H.P.; Software, J.G. and I.C.; Formal analysis, J.G., I.C., P.M. and H.P.; Investigation, J.G. and I.C.; Resources, P.M.; Writing—original draft, J.G. and I.C.; Writing—review & editing, P.M.; Supervision, H.P.; Funding acquisition, H.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding authors.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Abualigah, L.; Elaziz, M.A.; Khasawneh, A.M.; Alshinwan, M.; Ibrahim, R.A.; Al-qaness, M.A.; Mirjalili, S.; Sumari, P.; Gandomi, A.H. Meta-heuristic optimization algorithms for solving real-world mechanical engineering design problems: A comprehensive survey, applications, comparative analysis, and results. *Neural Comput. Appl.* **2022**, *34*, 4081–4110. [\[CrossRef\]](#)
2. Chaharmahali, G.; Ghandalipour, D.; Jasemi, M.; Molla-Alizadeh-Zavardehi, S. Modified metaheuristic algorithms to design a closed-loop supply chain network considering quantity discount and fixed-charge transportation. *Expert Syst. Appl.* **2022**, *202*, 117364. [\[CrossRef\]](#)
3. Martínez-Muñoz, D.; García, J.; Martí, J.; Yepes, V. Discrete swarm intelligence optimization algorithms applied to steel-concrete composite bridges. *Eng. Struct.* **2022**, *266*, 114607. [\[CrossRef\]](#)
4. Martínez-Muñoz, D.; García, J.; Martí, J.; Yepes, V. Deep learning classifier for life cycle optimization of steel–concrete composite bridges. *Structures* **2023**, *57*, 105347. [\[CrossRef\]](#)
5. García, J.; Lemus-Romani, J.; Altimiras, F.; Crawford, B.; Soto, R.; Becerra-Rozas, M.; Moraga, P.; Becerra, A.P.; Fritz, A.P.; Rubio, J.M.; et al. A binary machine learning cuckoo search algorithm improved by a local search operator for the set-union knapsack problem. *Mathematics* **2021**, *9*, 2611. [\[CrossRef\]](#)
6. Wolpert, D.H.; Macready, W.G. No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.* **1997**, *1*, 67–82. [\[CrossRef\]](#)

7. Kareem, S.S.; Mostafa, R.R.; Hashim, F.A.; El-Bakry, H.M. An effective feature selection model using hybrid metaheuristic algorithms for iot intrusion detection. *Sensors* **2022**, *22*, 1396. [\[CrossRef\]](#) [\[PubMed\]](#)
8. Yadav, R.K.; Mahapatra, R.P. Hybrid metaheuristic algorithm for optimal cluster head selection in wireless sensor network. *Pervasive Mob. Comput.* **2022**, *79*, 101504. [\[CrossRef\]](#)
9. García, J.; Leiva-Araos, A.; Crawford, B.; Soto, R.; Pinto, H. Exploring Initialization Strategies for Metaheuristic Optimization: Case Study of the Set-Union Knapsack Problem. *Mathematics* **2023**, *11*, 2695. [\[CrossRef\]](#)
10. Diana, R.O.M.; de Souza, S.R. Analysis of variable neighborhood descent as a local search operator for total weighted tardiness problem on unrelated parallel machines. *Comput. Oper. Res.* **2020**, *117*, 104886. [\[CrossRef\]](#)
11. Mirjalili, S.; Lewis, A. The whale optimization algorithm. *Adv. Eng. Softw.* **2016**, *95*, 51–67. [\[CrossRef\]](#)
12. Lai, X.; Hao, J.K.; Yue, D. Two-stage solution-based tabu search for the multidemand multidimensional knapsack problem. *Eur. J. Oper. Res.* **2019**, *274*, 35–48. [\[CrossRef\]](#)
13. Song, M.S.; Emerick, B.; Lu, Y.; Vasko, F.J. When to use Integer Programming Software to solve large multi-demand multidimensional knapsack problems: A guide for operations research practitioners. *Eng. Optim.* **2022**, *54*, 894–906. [\[CrossRef\]](#)
14. Cappanera, P.; Trubian, M. A local-search-based heuristic for the demand-constrained multidimensional knapsack problem. *INFORMS J. Comput.* **2005**, *17*, 82–98. [\[CrossRef\]](#)
15. Arntzen, H.; Hvattum, L.M.; Løkketangen, A. Adaptive memory search for multidemand multidimensional knapsack problems. *Comput. Oper. Res.* **2006**, *33*, 2508–2525. [\[CrossRef\]](#)
16. Gortazar, F.; Duarte, A.; Laguna, M.; Martí, R. Black box scatter search for general classes of binary optimization problems. *Comput. Oper. Res.* **2010**, *37*, 1977–1986. [\[CrossRef\]](#)
17. García, J.; Moraga, P.; Crawford, B.; Soto, R.; Pinto, H. Binarization Technique Comparisons of Swarm Intelligence Algorithm: An Application to the Multi-Demand Multidimensional Knapsack Problem. *Mathematics* **2022**, *10*, 3183. [\[CrossRef\]](#)
18. Cappanera, P.; Gallo, G.; Maffioli, F. Discrete facility location and routing of obnoxious activities. *Discret. Appl. Math.* **2003**, *133*, 3–28. [\[CrossRef\]](#)
19. Beaujon, G.J.; Marin, S.P.; McDonald, G.C. Balancing and optimizing a portfolio of R&D projects. *Nav. Res. Logist. NRL* **2001**, *48*, 18–40.
20. Avci, M.; Topaloglu, S. A multi-start iterated local search algorithm for the generalized quadratic multiple knapsack problem. *Comput. Oper. Res.* **2017**, *83*, 54–65. [\[CrossRef\]](#)
21. Sonuç, E.; Özcan, E. CUDA-based parallel local search for the set-union knapsack problem. *Knowl.-Based Syst.* **2024**, *299*, 112095. [\[CrossRef\]](#)
22. Wu, L.; Lin, K.; Lin, X.; Lin, J. List-Based Threshold Accepting Algorithm with Improved Neighbor Operator for 0–1 Knapsack Problem. *Algorithms* **2024**, *17*, 478. [\[CrossRef\]](#)
23. Lam, S.K.; Pitrou, A.; Seibert, S. Numba: A llvm-based python jit compiler. In Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC, Austin, TX, USA, 15 November 2015; pp. 1–6.
24. García, J.; Crawford, B.; Soto, R.; Castro, C.; Paredes, F. A k-means binarization framework applied to multidimensional knapsack problem. *Appl. Intell.* **2018**, *48*, 357–380. [\[CrossRef\]](#)
25. Kruskal, W.H.; Wallis, W.A. Use of ranks in one-criterion variance analysis. *J. Am. Stat. Assoc.* **1952**, *47*, 583–621. [\[CrossRef\]](#)
26. Dunn, O.J. Multiple comparisons using rank sums. *Technometrics* **1964**, *6*, 241–252. [\[CrossRef\]](#)
27. Mirjalili, S.; Mirjalili, S.M.; Lewis, A. Grey wolf optimizer. *Adv. Eng. Softw.* **2014**, *69*, 46–61. [\[CrossRef\]](#)

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.