Report for IT3052E Project

Optimization Algorithms for Order Picking-up route in Warehouse Large Size: Efficiency vs. Solution Quality

Nguyen T. Hieu[a], **Nguyen Q. Duong**[b], **Do X. Truong**[c] **and Mai G. Hy**[d]

[a]*Student's ID: 202416689*
[b]*Student's ID: 202416683*
[c]*Student's ID: 202416758*
[d]*Student's ID: 202400109*

Professor: Pham Q. Dung

Abstract—This study investigates the performance of various heuristic algorithms for solving a warehouse shelf selection problem, where the objective is to fulfill product demands with minimal travel distance and shelf usage. To enable systematic comparison, we developed a controlled test case generator that simulates realistic warehouse layouts and product distributions. Unlike exact optimization, our test case generator leverages a greedy strategy that incrementally selects shelves based on local criteria, such as distance or product availability. While this approach does not guarantee global optimality, it provides feasible, diverse, and scalable instances for algorithm benchmarking. We evaluate multiple heuristics across varying problem sizes (small, medium, large), reporting average runtime, total distance traveled, and number of shelves used. Additionally, each algorithm's performance is compared against the baseline greedy solution used during test case generation, offering insights into relative improvements and trade-offs between computation time and solution quality.

Keywords—*optimization, ortools, routing problems, algorithm*

1. Introduction

Warehouse management is a critical component in modern supply chain systems, where efficiency directly impacts operational cost and service quality. One fundamental subproblem in warehouse operations is the **shelf selection problem**, which involves determining a subset of shelves to visit in order to fulfill product demands with minimal resource usage. Key objectives typically include minimizing the number of shelves visited, reducing total travel distance, and ensuring timely order fulfillment.

This problem presents significant computational challenges due to its combinatorial nature. While exact methods such as Integer Programming (IP) can yield optimal solutions, they become impractical for large-scale instances due to prohibitive computation time. Therefore, **heuristic and metaheuristic approaches** are commonly employed to strike a balance between solution quality and runtime efficiency.

In this work, we conduct a comprehensive empirical comparison of several heuristic algorithms applied to the shelf selection problem. To facilitate a controlled and repeatable evaluation, we develop a **test case generator based on a greedy shelf selection strategy**. This generator simulates realistic warehouse layouts by randomly assigning products to shelves, generating demand vectors, and computing pairwise distances. Although the generator's solution is not globally optimal, it provides feasible and diverse instances against which other algorithms can be benchmarked.

Our experimental analysis focuses on three performance metrics: **runtime, number of shelves selected**, and **total travel distance**. Test cases are categorized into small, medium, and large scales to assess scalability and robustness. Moreover, all algorithms are evaluated not only in absolute terms but also **relative to the greedy-based baseline used in test generation**, enabling fair assessment of both gains and trade-offs.

Through this study, we aim to highlight the strengths and limitations of each heuristic under varying conditions, and to provide practical insights into their applicability for real-world warehouse optimization scenarios.

2. Problem Formulation

We consider a warehouse environment with a central depot (denoted as node 0) and M shelves (indexed from 1 to M). Each shelf j is located at a unique position in the warehouse. There are N product types, and shelf j may store a certain quantity of each product i, represented by the quantity matrix Q.

A warehouse worker starts at the depot (node 0), and needs to visit a subset of shelves to collect all products requested in a customer order. The total required quantity for each product i is given by the demand vector q. The worker must return to the depot after completing the pickups.

The goal is to find a visit sequence to shelves such that all demands are fulfilled, and the total travel distance (from depot $\rightarrow$ shelves $\rightarrow$ depot) is minimized. Each shelf can be visited at most once, and it is not necessary to visit all shelves.

Notation

- N: Number of product types.
- M: Number of shelves.
- $Q \in \mathbb{Z}_{\geq 0}^{N \times M}$: Quantity matrix where $Q_{i,j}$ denotes the number of units of product i available on shelf j.
- $q \in \mathbb{Z}_{>0}^{N}$: Demand vector where q_i is the required number of units for product i.
- $D \in \mathbb{R}^{(M+1)\times(M+1)}$: Distance matrix where $D_{i,j}$ is the distance between node i and node j, with node 0 being the depot.
- $x_j \in \{0, 1\}$: Binary decision variable indicating whether shelf j is selected (visited).
- $r_{i,j} \in \mathbb{Z}_{\geq 0}$: Number of units of product i retrieved from shelf j.

Objective Function

The primary objective is to minimize the total travel distance of the worker, starting and ending at the depot:

$$\min \text{TotalDistance} = D_{0,x_1} + \sum_{k=1}^{n-1} D_{x_k,x_{k+1}} + D_{x_n,0}$$

where $(x_1, x_2, \ldots, x_n)$ is the selected sequence of shelf visits. Alternatively, in simplified form with binary variables:

$$\min \sum_{j=1}^{M} x_j \cdot (D_{0,j} + D_{j,0})$$

Constraints

The following constraints must be satisfied:

1. **Demand satisfaction:**

$$\sum_{j=1}^{M} r_{i,j} \geq q_i \quad \forall i \in \{1, \ldots, N\}$$

2. **Shelf capacity:**

$$r_{i,j} \leq Q_{i,j} \cdot x_j \quad \forall i \in \{1, \ldots, N\}, \quad \forall j \in \{1, \ldots, M\}$$

3. **Visit consistency:**

$$x_j = 1 \iff \exists i \text{ such that } r_{i,j} > 0$$

This problem formulation leads to a mixed-integer optimization model. Due to the combinatorial nature of shelf selection and routing, exact methods become intractable at large scale. Thus, we adopt heuristic and hybrid strategies to approximate optimal solutions efficiently.

3. Methodology

3.1. Greedy Algorithm for Test Case Generation

To generate test cases efficiently and ensure that each instance is solvable, we employ a simple yet effective greedy selection algorithm. The key idea is to select shelves that offer the required products with the lowest estimated round-trip cost from the entrance.

Algorithm 1 GreedyShelfSelection(N, M, Q, D, q)

```
Initialize selected_shelves ← ∅
Initialize remaining_demand ← q
for each product i = 1 to N do
    candidates ← [ ]
    for each shelf j = 1 to M do
        if Q[i][j] > 0 then
            c ← D[0][j + 1] + D[j + 1][0]
            Append (c, j) to candidates
        end if
    end for
    Sort candidates by c in ascending order
    for each (c, j) in candidates do
        if remaining_demand[i] = 0 then
            break
        end if
        t ← min(Q[i][j], remaining_demand[i])
        remaining_demand[i] −= t
        selected_shelves ← selected_shelves ∪ {j}
    end for
end for
return sorted list of selected_shelves
```

This method was chosen due to the following advantages:

- **Speed**: Runs in $O(NM \log M)$ time, making it well-suited for large-scale test generation.
- **Feasibility**: Ensures that each generated instance satisfies all demand constraints if possible.
- **Distance-aware**: Tends to favor shelves with lower round-trip cost from the entrance, producing realistic and meaningful layouts.

3.2. Heuristic-based Algorithms

Greedy

The `Greedy` algorithm incrementally selects shelves to fulfill product demands by always visiting the nearest unvisited shelf that contains at least one needed product. The main idea is to minimize the immediate travel cost while greedily covering as many product requirements as possible.

Algorithm Description. Let the worker start at the warehouse entrance (position 0). At each step, the algorithm scans all shelves that:

- have not been visited yet, and
- contain at least one unit of a product that is still required.

Among those, it selects the shelf with the minimum distance from the current position. After visiting a shelf, it updates the remaining demand vector and moves the current position to that shelf. This process repeats until all product requirements are satisfied.

Algorithm 2 GreedyShelfRouting(N, M, Q, D, q)

```
pos ← 0                                          ▷ Start at the entrance
selected ← [ ], visited ← ∅
remaining ← q
while any remaining[i] > 0 do
    candidates ← {j ∉ visited | ∃i : Q[i][j] > 0 ∧ remaining[i] > 0}
    if candidates = ∅ then
        raise error ("No feasible solution")
    end if
    next ← arg min_{j∈candidates} D[pos][j]
    selected.append(next), visited.add(next)
    for i = 1 to N do
        taken ← min(Q[i][next], remaining[i])
        remaining[i] −= taken
    end for
    pos ← next
end while
return |selected|, selected
```

Remarks. This greedy algorithm is:

- **Fast**: Each step selects the next shelf with minimal overhead, leading to overall complexity of roughly $O(NM^2)$ in worst-case.
- **Feasible**: Always attempts to reduce the remaining demand and guarantees termination if a solution exists.
- **Shortsighted**: May not find the globally optimal tour as it optimizes only the next move.

Nonetheless, it serves as a strong baseline due to its simplicity and ability to generate quick, valid solutions in practical scenarios.

Local Search with 2-Opt

This method combines a fast greedy route construction with local refinement. The goal is to select shelves that satisfy product demand while minimizing travel distance.

Method.

1. Use **Nearest Neighbor** to construct an initial route from the warehouse door (node 0), selecting the closest shelf that helps fulfill demand.
2. Stop once all product demands are satisfied, then return to the door.
3. Apply **2-opt** to iteratively swap path segments to reduce total distance.

Algorithm 3 LocalSearchTwoOpt(Q, D, q)

```
1: path ← NearestNeighbor(Q, D, q)
2: path ← TwoOpt(path, D)
3: return shelves in path excluding depot (0)
```

NearestNeighbor: Greedily select the closest unvisited shelf until all product demands are met.

TwoOpt: For each pair of non-overlapping edges in the route, reverse the segment if the total distance is reduced. Repeat until no improvement.

Remark. This hybrid approach balances efficiency and solution quality, making it suitable for moderately large problem instances where exact methods are too slow.

Greedy Local Search

This method iteratively selects the nearest unvisited shelf that can contribute to unmet product demands. It terminates when all product demands are satisfied.

Algorithm 4 GreedyLocalSearch(Q, D, q)

```
remaining ← q, pos ← 0, visited ← ∅, route ← [ ]
while any demand remains in remaining do
    C ← unvisited shelves that provide remaining products
    j* ← arg min_{j∈C} D[pos][j]
    Add j* to route, mark as visited
    Update remaining by subtracting picked products from shelf j*
    pos ← j*
end while
return route
```

Remark. This algorithm is simple, efficient, and effective in practice for generating feasible solutions quickly. It is often used as a baseline or initializer for more advanced heuristics.

3.3. Dynamic Programming-enhanced Algorithms

DP-Greedy

This hybrid algorithm is designed to efficiently solve the warehouse shelf selection problem by combining two complementary strategies:

- **Exact Dynamic Programming (DP)** for smaller problem instances, where the number of shelves M is small enough (specifically, $M \leq 18$) to allow an exhaustive search of all subsets of shelves. This guarantees an optimal solution in these cases.
- **Greedy Heuristic** as a fallback for larger instances, where the exact DP becomes computationally infeasible due to exponential growth in the number of subsets. The greedy heuristic provides a fast, though not necessarily optimal, approximation.

Algorithm 5 DP_Greedy(Q, D, q)

```
if M ≤ 18 then
    Exact DP Phase:
    for all subsets S ⊆ {1, 2, ..., M} do
        Compute total supply vector ∑Q[S] by summing quantities from shelves in S
        if for every product, total supply ≥ demand q then
            Run DP to find the minimal travel cost route visiting shelves in S
            if DP finds a feasible route then
                return this optimal route
            end if
        end if
    end for
end if
Fallback Greedy Phase:
Initialize current position at warehouse entrance (node 0)
Initialize set of shelves to visit as unvisited shelves U = {1, ..., M}
Initialize supply collected as zero vector for all products
while collected supply does not satisfy demand q do
    Select the nearest shelf s ∈ U to current position that can contribute to unmet demand
    Move to shelf s, update collected supply and mark s as visited (remove from U)
    Update current position to s
end while
return the constructed route visiting selected shelves in order
```

Remarks.

- The **Exact DP phase** fully enumerates subsets of shelves and uses dynamic programming to determine the shortest route visiting all shelves in the subset. This ensures the optimality of the solution but has exponential time complexity in M.
- The **Greedy fallback** phase is a practical heuristic that constructs a route by iteratively visiting the nearest shelf that helps fulfill the remaining demand, offering a scalable solution for larger problem sizes.
- By combining these two phases, the algorithm balances **optimality and efficiency** — using the exact solution when possible and gracefully degrading to a heuristic approach when necessary.

DP-Greedy with 2-Opt

This method extends the **DP-Greedy** approach by incorporating a local search optimization step known as *2-opt* during the heuristic fallback phase. The 2-opt procedure iteratively refines the constructed greedy route by swapping pairs of edges to reduce total travel distance, often yielding significantly improved solutions without a large increase in computation time.

Algorithm 6 DP-Greedy with 2-Opt(Q, D, q)

```
if M ≤ 18 then
    Exact DP Phase:
    for all subsets S ⊆ {1, ..., M} do
        Compute total supply ∑Q[S] from shelves in S
        if supply covers demand q then
            Use DP to find minimal-cost route visiting shelves in S
            if feasible solution found then
                return optimal DP route
            end if
        end if
    end for
end if
Greedy Heuristic Phase:
Initialize current location at entrance (node 0)
Initialize unvisited shelves U = {1, ..., M}
Initialize collected supply as zero vector
while collected supply is insufficient to meet q do
    Select nearest unvisited shelf s ∈ U that contributes to unmet demand
    Move to shelf s, update collected supply and remove s from U
    Update current location to s
end while
Construct route P = [0, s_1, s_2, ..., s_k]
2-Opt Local Search Phase:
Initialize improvement flag as True
while improvement flag is True do
    Set improvement flag to False
    for i = 1 to k − 2 do
        for j = i + 1 to k do
            if swapping edges (s_{i−1}, s_i) and (s_j, s_{j+1}) reduces total distance then
                Reverse segment between s_i and s_j in route P
                Set improvement flag to True
            end if
        end for
    end for
end while
return improved route P excluding the entrance node 0
```

Remarks.

- The **Exact DP phase** attempts to find the optimal solution for

small instances by enumerating subsets of shelves and computing shortest routes.
- The **Greedy phase** quickly constructs a feasible route by iteratively visiting the nearest shelf that satisfies unmet demand.
- The **2-Opt local search** iteratively improves the greedy route by swapping pairs of edges to reduce travel distance, which tends to enhance solution quality significantly with minimal computational overhead.
- Overall, this hybrid method balances exact optimization and heuristic efficiency, producing better solutions for larger problems than the pure greedy heuristic alone.

4. Experiment

4.1. Experimental Setup

We conduct extensive experiments to evaluate the performance of five algorithms, including our DP-Greedy variants and several baselines, on warehouse shelf selection problems of varying sizes.

Hardware and Environment All experiments were run on a machine equipped with Apple M3 Pro chip and 18GB RAM. The algorithms are implemented in Python 3.12.7.

Test Data Generation Test cases are synthetically generated using a dedicated Python script, controlled by a Bash wrapper script that produces 10 test cases for each size category: *small*, *medium*, and *large*. The parameters for each category are:

- Number of products N: 10 (small), 40 (medium), 100 (large)
- Number of shelves M: 20 (small), 100 (medium), 400 (large)
- Maximum quantity per shelf: 20 (small), 100 (medium), 200 (large)
- Number of shelves to visit for fulfilling demand: 6 (small), 20 (medium), 40 (large)

The generated test data includes the supply matrix Q, distance matrix D, and demand vector q. Ground-truth expected outputs for some algorithms (e.g., Greedy) are also stored for evaluation. A sparsity level of 0.4 is applied to control the number of shelves stocking each product, simulating realistic product distribution. Demand values are scaled using a spread factor of 2.0, ensuring that each product's demand is feasible relative to its total supply but non-trivial to satisfy.

4.2. Algorithms Evaluated

In this study, we compare the performance of five aforementioned algorithms for the warehouse shelf selection and routing problem:

- **Exact Dynamic Programming (Exact DP):** This method finds the optimal solution by enumerating all subsets of shelves up to size $M \leq 18$, guaranteeing minimal travel cost while satisfying demand constraints.
- **DP-Greedy:** A hybrid approach that uses exact DP when $M \leq 18$, otherwise falls back to a greedy nearest-neighbor heuristic that iteratively selects the closest unvisited shelf until demand is met.
- **DP-Greedy with 2-Opt:** Extends DP-Greedy by applying a local search improvement technique (2-opt) on the greedy route to reduce travel distance by swapping edges.
- **Greedy:** A pure heuristic that starts from the entrance and iteratively visits the nearest unvisited shelf, stopping when the accumulated supply meets the required demand.
- **Greedy with 2-Opt:** Enhances the Greedy heuristic by applying 2-opt local search to optimize the initial route.

4.3. Evaluation Metrics

We measure and compare algorithms based on:

- **Total travel distance:** Sum of distances along the selected route
- **Runtime:** Wall-clock time in seconds
- **Number of shelves visited:** Route length excluding the depot
- **Feasibility:** Whether the solution meets the demand q

4.4. Experiment Procedure

For each test case size, we run all algorithms on all generated instances and record their outputs. We ensure that all solutions satisfy the demand constraints. Runtime is measured using Python's `time.time()` function.

5. Results

In this section, we present a comprehensive evaluation of all five algorithms on three test sizes: **small**, **medium**, and **large**. Each configuration was tested with 10 randomized instances per size.

We evaluate performance based on three key metrics:

- **Runtime (s)** – computation time measured from input loading to output writing.
- **Shelves Visited** – the number of shelves selected to satisfy the demand vector.
- **Total Travel Distance** – sum of distances in the selected route.

5.1. Runtime Analysis

Figure 1 presents the average runtime of each algorithm across three problem sizes: small, medium, and large. All methods exhibit low runtime on small instances, typically under 0.003 seconds. As the problem size increases, runtime differences become more pronounced. `DP-Greedy` consistently demonstrates the lowest runtime across all sizes, scaling efficiently even for large problems (around 0.005 seconds), significantly outperforming the other methods.

In contrast, algorithms incorporating local search or 2-opt heuristics—such as `Greedy_Local_Search`, `Local_Search_Two_Opt`, and `DP_Greedy_Two_Opt`—exhibit much higher runtimes on large instances, ranging from 0.034 to 0.048 seconds. This represents up to a 9-fold increase compared to `DP-Greedy`. These methods, while potentially offering improved solution quality, incur a substantial computational cost.

The plain `Greedy` algorithm shows moderate runtime, slightly higher than `DP-Greedy`, but remains within a reasonable range. Overall, greedy-based methods are consistently fast, and `DP-Greedy` in particular achieves an excellent balance between scalability and efficiency, making it highly suitable for large-scale applications where speed is critical.

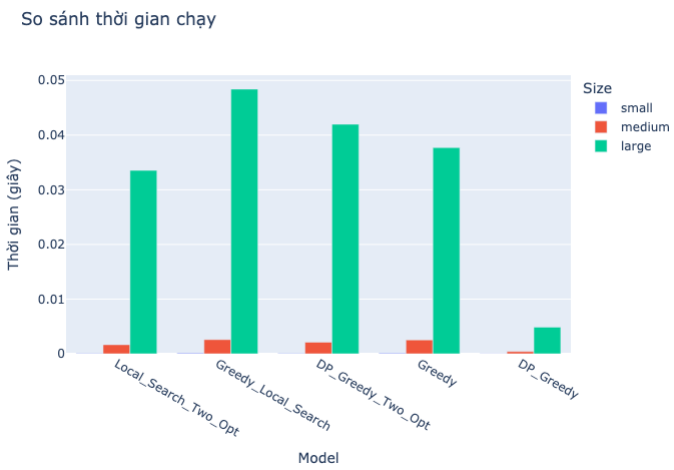

Figure 1. Average runtime of each algorithm on small, medium, and large test cases.

5.2. Shelves Visited

Figure 2 reports the total number of shelves selected by each algorithm across different problem sizes. Overall, all methods return a comparable number of shelves, with small variations across algorithms. Notably, `DP-Greedy` and its variants (`DP_Greedy_Two_Opt`) consistently achieve slightly fewer shelves across all sizes, suggesting better global planning. For example, on large instances, they use marginally fewer shelves than the other methods, indicating improved packing efficiency.

In contrast, pure greedy approaches—such as `Greedy` and `Greedy_Local_Search`—tend to select slightly more shelves, particularly on medium and large inputs. This is likely due to their locally optimal decisions, which may lead to suboptimal global layouts. Meanwhile, hybrid methods incorporating local search or 2-opt refinements (e.g., `Local_Search_Two_Opt`) offer a modest improvement over greedy baselines but do not outperform `DP-Greedy`.

Overall, while the absolute differences in shelf count are small, the consistent advantage of `DP-Greedy` highlights its potential in optimizing not only runtime but also solution quality in terms of space usage.

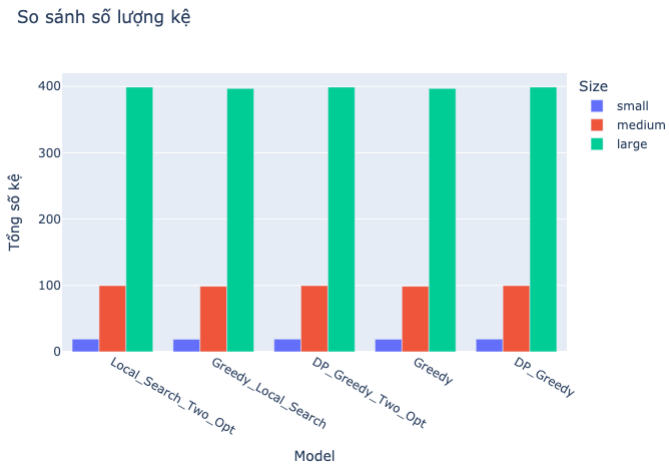

Figure 2. Average number of shelves selected to satisfy demand across test sizes.

5.3. Total Distance Traveled

Figure 3 compares the total travel distance for each algorithm across problem sizes. As expected, methods incorporating the `2-Opt` heuristic—namely `DP_Greedy_Two_Opt` and `Local_Search_Two_Opt`—consistently achieve shorter total distances than their respective baselines. This effect is particularly pronounced on large instances, where `DP_Greedy_Two_Opt` achieves the lowest overall distance among all methods, suggesting effective route refinement.

In contrast, both `Greedy` and `DP-Greedy` show higher total distances, with nearly identical performance to each other. The inclusion of local search significantly improves upon these, especially when combined with `2-Opt`. For instance, `Greedy_Local_Search` achieves noticeably shorter distances than plain `Greedy`, though still not as low as the full `2-Opt` variants.

These results confirm the benefit of post-processing heuristics in improving routing efficiency. While greedy-based methods are fast, integrating local refinement—particularly `2-Opt`—can substantially reduce travel cost, especially for larger-scale instances.

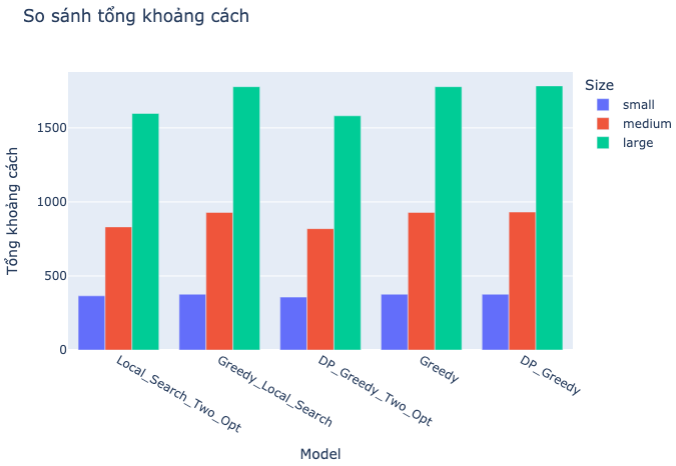

Figure 3. Average total travel distance by algorithm for each test size.

5.4. Tabular Summary

Table 1. Average Runtime (seconds) by Algorithm and Problem Size

Algorithm	Small	Medium	Large
Local_Search_Two_Opt	0.000068	0.001651	0.033545
Greedy_Local_Search	0.000131	0.002585	0.048416
DP_Greedy_Two_Opt	0.000069	0.002111	0.041992
Greedy	0.000123	0.002526	0.037710
DP_Greedy	0.000032	0.000405	0.004863

Runtime (seconds)

Table 2. Average Number of Shelves Visited by Algorithm and Problem Size

Algorithm	Small	Medium	Large
Local_Search_Two_Opt	19	100	399
Greedy_Local_Search	19	99	397
DP_Greedy_Two_Opt	19	100	399
Greedy	19	99	397
DP_Greedy	19	100	399

Shelves Visited

Table 3. Average Total Travel Distance by Algorithm and Problem Size

Algorithm	Small	Medium	Large
Local_Search_Two_Opt	365.5	831.3	1597.3
Greedy_Local_Search	376.7	929.1	1777.8
DP_Greedy_Two_Opt	357.1	819.4	1581.2
Greedy	376.7	929.1	1777.8
DP_Greedy	376.6	931.5	1783.3

Total Travel Distance

6. Conclusion

This study presented a comparative analysis of five algorithms for the shelf selection and routing problem: `Greedy`, `Greedy + 2-Opt`, `DP-Greedy`, `DP-Greedy + 2-Opt`, and a pure `2-Opt` local search. The experiments were conducted on synthetic datasets of varying sizes (Small, Medium, Large), with evaluation criteria focused on

runtime efficiency, the number of shelves visited, and total travel distance.

Key Findings.

- **Runtime:** `DP-Greedy` achieved the fastest execution across all test sizes, significantly outperforming other methods especially in the large case (`0.00486`s vs. `0.04`s for other heuristics). This is expected since it uses exact dynamic programming only on small instances and falls back to a minimal greedy core for larger inputs.
- **Shelf Coverage:** All algorithms visit roughly the same number of shelves across all test sizes (e.g., around 100 shelves in medium and 399–400 in large). This confirms that they all manage to meet the supply demand constraint without over-picking.
- **Route Quality:** In terms of total travel distance, `DP-Greedy + 2-Opt` achieved the best results across all test sizes, consistently outperforming other methods. For example, in the large case, it produced routes about 200 units shorter than naive `Greedy`, demonstrating the benefit of combining dynamic programming-based shelf selection with local path optimization.

Conclusion. Overall, `DP-Greedy + 2-Opt` emerges as the most effective approach, striking a strong balance between computational efficiency and solution quality. It retains the fast runtime of `DP-Greedy` while substantially improving the route quality through local search. For practical deployment in warehouse systems, this hybrid method is highly recommended due to its scalability, simplicity, and robustness.

Reproducibility The full source code, dataset generator, and experiment scripts are available at: https://github.com/hieunguyen-cyber/Order-Picking-up-route-in-Warehouse-Large-Size