WarehouseBot: Pick-and-Drop Optimization

Apurv Kushwaha
University of Minnesota Twin Cities
kushw064@umn.edu

December 18, 2025

Abstract

Efficient routing is a key challenge in automated warehouses, where robots must pick up and deliver multiple parcels while minimizing travel distance. This project studies a single-robot pick-and-drop optimization problem with precedence constraints, where each parcel must be picked up before it is delivered. The problem is addressed using a two-layer approach: A* search for grid-based path planning and local search algorithms for task sequencing. A greedy baseline, Hill Climbing with restarts, and Simulated Annealing are implemented and compared. Experimental results on both static and randomly generated warehouse layouts show consistent improvements over the greedy approach, with typical mean improvements around 5–12% depending on instance size and parcel count, with Simulated Annealing achieving the best solution quality at the cost of increased runtime.

1 Introduction

Autonomous robots are widely used in modern e-commerce warehouses to reduce human labor and improve efficiency. One fundamental challenge in such systems is determining the order in which a robot should visit multiple pickup and delivery locations while minimizing total travel distance. This project addresses that challenge for a single warehouse robot operating in a grid-based environment with obstacles.

Each parcel in the warehouse is associated with a pickup location and a corresponding drop location. A key constraint is that every pickup must occur before its associated delivery. This makes the problem more complex than the classical Traveling Salesperson Problem, as not all permutations of locations are feasible. For larger instances, exhaustive search becomes impractical, motivating the use of heuristic and local search techniques.

The proposed system decomposes the problem into two layers. First, geometric path planning is handled using A* search to compute shortest paths between all relevant locations. Second, a task sequencing layer uses local search algorithms to determine an efficient visiting order that respects precedence constraints. This project integrates classical search and local search optimization techniques and evaluates their performance through systematic experiments, making it significantly more complex than earlier course programming assignments.

2 Related Work

Zhou and Li [1] studied routing optimization for intelligent vehicles in automated warehouses, focusing on multi-vehicle coordination using evolutionary techniques. Wang and Chen [2] compared constructive heuristics for logistics robots and demonstrated that greedy approaches often perform poorly when precedence constraints are present. Roodbergen and De Koster [3] applied TSP-based methods to order picking in warehouses, showing the importance of efficient routing algorithms in practical logistics scenarios.

More recent work by Msala et al. [4] proposed integrated frameworks combining the Hungarian algorithm with TSP for multi-robot coordination in heterogeneous systems. Sung and Jeong [5] developed evolutionary algorithms specifically for TSP with precedence constraints, demonstrating the effectiveness of adaptive genetic operators.

In contrast, this project focuses on the single-robot case and emphasizes a clear separation between path planning and task sequencing. This modular design enables direct comparison of different local search strategies while keeping the system simple and interpretable.

3 System Overview

The WarehouseBot system follows a two-layer architecture. In the first layer, the warehouse is modeled as a two-dimensional grid with obstacles representing shelves. A* search is used to compute shortest paths between the robot start location, all pickup locations, and all drop locations. The resulting distances are stored in a cost matrix.

In the second layer, the cost matrix is used as input to a task sequencing module. This module searches for a minimum-cost route that visits all required locations while ensuring that each pickup occurs before its corresponding drop. By separating geometric planning from combinatorial optimization, the system remains modular and easier to evaluate.

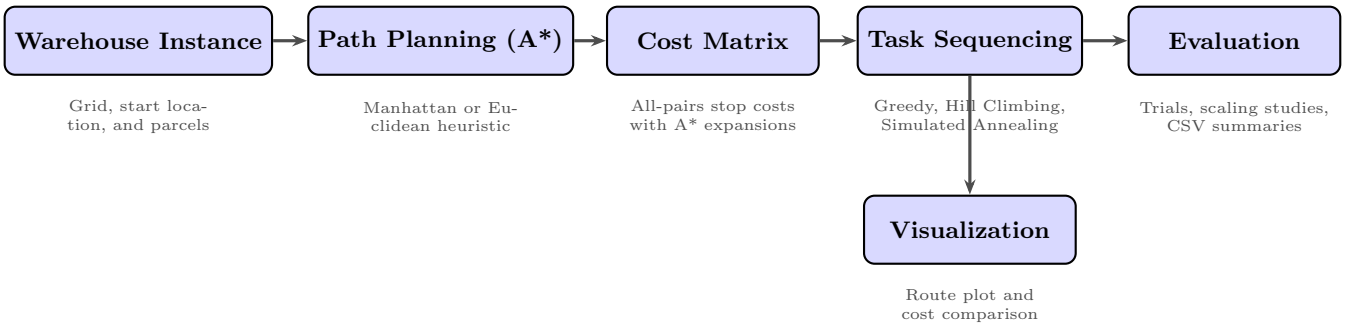

Figure 1: WarehouseBot pipeline: path planning produces a cost matrix, which enables precedence-constrained sequencing and evaluation.

4 Path Planning Layer

Path planning is performed using the A* search algorithm on a grid with four-directional movement and unit step cost. Each grid cell represents either free space or an obstacle. A* maintains a priority queue ordered by the evaluation function $f(n) = g(n) + h(n)$, where $g(n)$ is the cost from the start node and $h(n)$ is a heuristic estimate to the goal.

Two admissible heuristics were implemented: Manhattan distance and Euclidean distance. While both guarantee optimal paths, Manhattan distance is more appropriate for grid-based movement and consistently expanded fewer nodes in practice. For a problem with n parcels, we compute an all-pairs cost matrix over the $2n + 1$ stops (depot + pickups + drops). Because the grid is undirected with unit costs, costs are symmetric and we exploit this by computing only unique pairs and mirroring the results.

5 Task Sequencing Layer

The task sequencing layer solves a precedence-constrained routing problem using the cost matrix produced by path planning.

5.1 Problem Formulation

A route is represented as an ordered list of locations starting from the depot. For n parcels, the route includes the depot, n pickup locations, and n corresponding drop locations. A route is considered feasible if each pickup appears before its associated drop. The cost of a route is computed as the sum of travel costs between consecutive locations in the route.

5.2 Greedy Baseline

The greedy algorithm constructs a route incrementally by repeatedly selecting the nearest feasible unvisited location. Drop locations only become eligible once their corresponding pickup has been visited. Although

this approach is fast and simple, it frequently leads to suboptimal solutions due to locally optimal but globally poor decisions.

5.3 Hill Climbing with Restarts

Hill Climbing improves an initial solution by iteratively exploring neighboring routes generated through swaps and insertions. Only neighbors that maintain feasibility are considered. The algorithm accepts an improving neighbor and terminates when no better neighbor is found. To reduce sensitivity to local minima, multiple random restarts are used, and the best solution across all restarts is selected.

5.4 Simulated Annealing

Simulated Annealing extends Hill Climbing by allowing occasional acceptance of worse solutions. A worse move is accepted with probability $e^{-\Delta/T}$, where Δ is the increase in cost and T is a temperature parameter. The temperature decreases gradually over time according to an exponential cooling schedule. This mechanism allows the search to escape local minima and generally produces higher-quality solutions at the expense of additional computation.

6 Experimental Setup

The system was evaluated on both static and randomly generated warehouse layouts. Experiments varied grid size, obstacle density, and number of parcels to test robustness and scalability. Performance was measured using total route cost, percentage improvement over the greedy baseline, runtime, and A* node expansions. Multiple experiment suites were conducted, including baseline evaluations, scaling studies, parameter sweeps for Simulated Annealing, ablation studies, and stress tests on larger instances.

7 Results and Analysis

Across all experiments, Hill Climbing and Simulated Annealing consistently improved over the greedy baseline, though the magnitude varied with grid size, obstacle density, and number of parcels. On the primary benchmark setting (15×15 grid, 20% obstacles, 5 parcels, 30 trials), Hill Climbing achieved about a 10% mean improvement over greedy, while Simulated Annealing achieved about an 8% mean improvement. Both methods increased runtime compared to greedy, but had similar runtimes in our implementation. Overall, Hill Climbing tended to provide the best average cost across trials, while Simulated Annealing remained competitive and occasionally produced the best solution on individual instances.

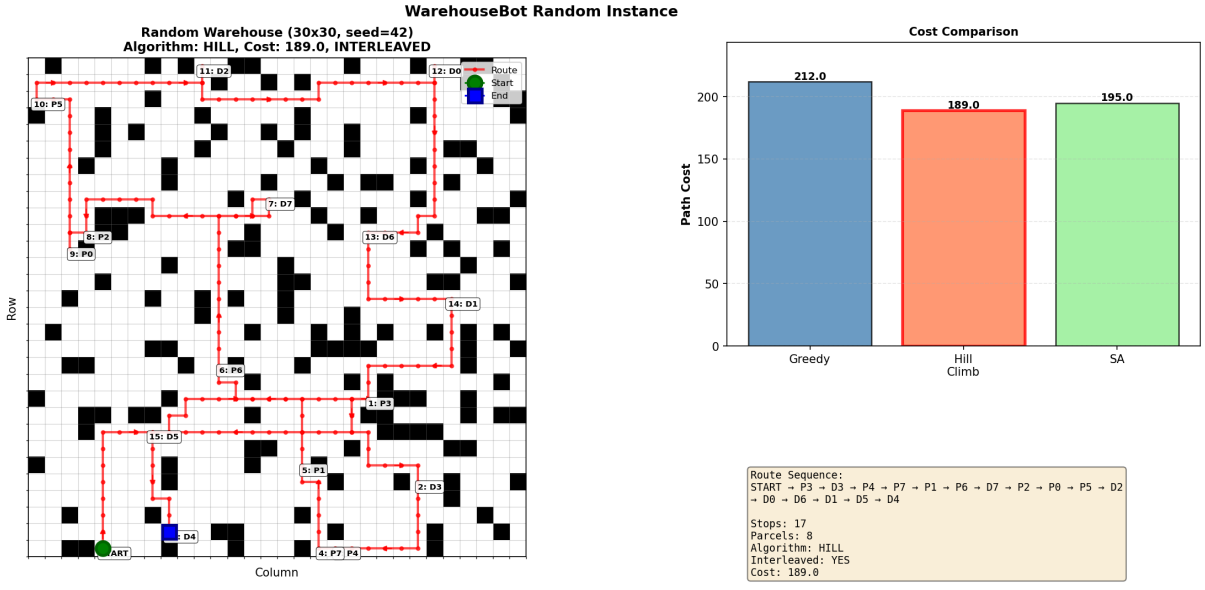

Figure 2: Example random warehouse instance (30×30, 20% obstacles, 8 parcels; 17 stops). The optimized route is interleaved, meaning the robot performs some drop-offs before completing all pickups, while still respecting precedence constraints (each Pi occurs before Di).

Scaling experiments showed that problem difficulty increases with grid size and parcel count. On larger instances, both Hill Climbing and Simulated Annealing continued to outperform greedy, with performance varying by map and parameter settings. The Manhattan heuristic consistently expanded fewer nodes than Euclidean during path planning, confirming its suitability for 4-neighbor grid movement.

8 Discussion and Conclusion

This project demonstrates that local search methods are effective for optimizing warehouse robot routing under precedence constraints. The two-layer decomposition successfully separates geometric path planning from combinatorial task sequencing, enabling clear analysis and comparison of algorithms.

The results highlight several key insights: greedy heuristics are insufficient for constrained routing problems, Hill Climbing with restarts offers a strong balance between solution quality and runtime, Hill Climbing with restarts provided the best average cost across trials, while Simulated Annealing sometimes found the best solutions on individual instances. Limitations of the current system include the single-robot assumption, static environments, and simplified movement model.

Future work could extend this system to multi-robot coordination, dynamic replanning in changing environments, or alternative optimization techniques such as genetic algorithms or constraint programming. Overall, the project achieves its goals by implementing a complete optimization pipeline, empirically evaluating multiple algorithms, and demonstrating meaningful improvements over baseline approaches.

Software Requirements and Instructions

Software Requirements

The implementation requires Python 3.8+ on Linux/macOS/Windows.

- `numpy` (matrix operations and numerical computations)
- `matplotlib` (visualization and plots)

All experiments are deterministic given a fixed seed (default seed = 42), and do not require external datasets.

All code is provided in the submitted `warehousebot.zip` file and is also available at the project repository: https://github.com/ApurvK032/WarehouseBot-Pick-and-Drop-Optimization.git

Instructions to Run

Navigate to the project root directory containing `warehousebot/` and execute the following commands:

Basic Evaluation:

```
python3 -m warehousebot.evaluation
```

This runs the static demo instance and 30 random trials, printing performance statistics.

Generate Visualization:

```
python3 -m warehousebot.visualize --save figure.png
```

This creates a figure showing the warehouse grid, robot path, and algorithm comparison charts.

Additional Experiments:

```
python3 -m warehousebot.evaluation --scaling    # Scaling study
python3 -m warehousebot.evaluation --sa-sweep   # Parameter sweep
python3 -m warehousebot.evaluation --ablation   # Ablation study
python3 -m warehousebot.evaluation --stress     # Stress tests
```

Large-Instance Experiments (Higher Complexity):

```
python3 -m warehousebot.evaluation --scaling
python3 -m warehousebot.evaluation --stress
```

The scaling study evaluates multiple grid sizes and parcel counts (e.g., 10×10 to 20×20 with varying obstacle rates). The stress test evaluates large scenarios such as 25×25 with 7 parcels and 30×30 with 8 parcels to demonstrate scalability.

All experiments produce console output with summary statistics. Some experiment modes also export CSV summaries (e.g., `results_trials.csv`, `scaling_study.csv`, and `ablation_study.csv`) for reproducible reporting. Detailed usage information is provided in the included `README.md` file.

References

[1] Zhou, X., Li, Y., Dong, Y., and Gu, J. (2014). Routing Optimization of Intelligent Vehicle in Automated Warehouse. *Discrete Dynamics in Nature and Society*, 2014, Article ID 789754. https://doi.org/10.1155/2014/789754

[2] Wang, H., and Chen, W. (2021). Task Scheduling for Transport and Pick Robots in Logistics: A Comparative Study on Constructive Heuristics. *Autonomous Intelligent Systems*, 1(17). https://doi.org/10.1007/s43684-021-00017-9

[3] Roodbergen, K. J., and De Koster, R. (2001). Routing Order Pickers in a Warehouse with a Middle Aisle. *European Journal of Operational Research*, 133(1), 32-43. https://doi.org/10.1016/S0377-2217(00)00177-6

[4] Msala, Y., Hamed, O., Talea, M., and Aboulfatah, M. (2024). A Novel Method for Enhancing Warehouse Operations Using Heterogeneous Robotic Systems for Autonomous Pick-and-Deliver Tasks. *EAI Endorsed Transactions on AI and Robotics*, 3(6). https://publications.eai.eu/index.php/airo/article/view/9913

[5] Sung, J., and Jeong, B. (2014). An Adaptive Evolutionary Algorithm for Traveling Salesman Problem with Precedence Constraints. *The Scientific World Journal*, 2014, Article ID 313767. https://doi.org/10.1155/2014/313767